读者注意：本文档内容仅限于说明1.280版本的设计与实现情况，易达保留在后续版本中更改设计和实现方式的权利。

1. 快速入门

本章将通过一个简单的策略程序，向投资者演示接入并使用易达柜台的必要步骤，帮助投资者建立易达API的基本概念。

1.1. 环境准备

首先，我们需要从https://www.hanlinit.com/download/ydclient_1_280_78_0/下载API，下载前请先注册官网投资者账号，待通过审批后即可下载（我们承诺该信息仅用于发布与易达有关信息，不用于其他用途）。下载完成后，请解压ydClient_1_280_78_0.tgz文件，我们会使用ydClient/win64和ydAPI目录中的工具和程序。

接下来我们先通过易达的Windows GUI客户端ydClient直接连接柜台，通过ydClient我们能方便检查后续编写的程序的运行结果。首先进入ydClient/win64目录，将YDConfig.ini修改为以下内容：

```
##########################################
######### Network configurations  #########
##########################################

# IP address of yd trading server
# TCP port of yd trading server, other ports will be delivered after logged in

TradingServerIP=118.190.175.212
TradingServerPort=41500

##########################################
######### Trading configurations  #########
##########################################

# Choose trading protocol, optional values are TCP, UDP, XTCP
TradingProtocol=UDP

# Timeout of select() when receiving order/trade notifications, in millisec. -1 indicates running without select()
TradingServerTimeout=10

# Affinity CPU ID for thread to send TCP trading and receive order/trade notifications, -1 indicate no need to set CPU affinity
TCPTradingCPUID=-1

# Affinity CPU ID for thread to send XTCP trading, -1 indicate no need to set CPU affinity
XTCPTradingCPUID=-1

##########################################
```

```
28 ####### MarketData configurations  ########
29 ##########################################
30
31 # Whether need to connect to TCP market data server
32 ConnectTCPMarketData=yes
33
34 # Timeout of select() when receiving TCP market data, in millisec. -1
   indicates running without select()
35 TCPMarketDataTimeout=10
36
37 # Affinity CPU ID for thread to receive TCP market data, -1 indicate no
   need to set CPU affinity
38 TCPMarketDataCPUID=-1
39
40 # Whether need to receive UDP multicast market data
41 ReceiveUDPMarketData=no
42
43 ##########################################
44 ####### Misc configurations        ########
45 ##########################################
46
47 AppID=yd_dev_1.0
48 AuthCode=ecbf8f06469eba63956b79705d95a603
```

上述配置信息指向是一套易达提供的上期所、能源所互联网测试环境，该环境7*24小时运行，会播放某一特定交易日的生产行情，方便投资者调试程序，详情请参见https://www.hanlinit.com/docs/dev-environments/。

配置完成后，请双击运行YDClient.exe，并使用001-100中的任一账户（密码同用户名）登陆，登陆成功后可以浏览各个菜单中的内容。

完成上述步骤后，我们将开始准备我们的策略程序。

1.2. 策略程序

要编译出可以运行的程序，需要先准备好易达程序的编译环境。请将ydAPI/linux64、ydAPI/linclude以及ydAPI/example下的所有文件都拷贝到Linux服务器的工作文件夹下面，并使用下列命令编译：

```
g++ -fpic -g -std=c++11 -c -O3 -pthread -Wall -c ydExample.cpp -o
ydExample.o
g++ -fpic -g -std=c++11 -c -O3 -pthread -Wall c Example1.cpp -o Example1.o
g++ -fpic -g -std=c++11 -c -O3 -pthread -Wall -c Example2.cpp -o
Example2.o
g++ -fpic -g -std=c++11 -c -O3 -pthread -Wall -c Example3.cpp -o
Example3.o
g++ -fpic -g -std=c++11 -c -O3 -pthread -Wall -c Example4.cpp -o
Example4.o
g++ -fpic -g -std=c++11 -c -O3 -pthread -Wall -c Example5.cpp -o
Example5.o
g++ -fpic -g -std=c++11 -c -O3 -pthread -Wall -c Example6.cpp -o
Example6.o
g++ -fpic -g -std=c++11 -c -O3 -pthread -Wall -c Example7.cpp -o
Example7.o
g++ -fpic -g -std=c++11 -c -O3 -pthread -Wall -c Example8.cpp -o
Example8.o
g++ -g -std=c++11 -o ydExample ./ydExample.o ./yd.so ./Example1.o
./Example2.o ./Example3.o ./Example4.o ./Example5.o ./Example6.o
./Example7.o ./Example8.o -m64 -Wall -lpthread -lrt -ldl
```

编译成功后，将上述拷贝到YDConfig.ini的内容同样拷贝到config.txt文件中，然后就可以运行第一个样例程序了，其中username和password使用登陆ydClient的用户名和密码替换：

```
# ./ydExample <example name> <config file> <username> <password>
<instrumentID>
./ydExample Example1 config.txt <username> <password> cu2203
```

当出现以下信息时，即表示策略程序开始执行，其中的告警信息请参考信息采集中关于dmidecode的内容：

```
sudo: a password is required
login successfully
Position=0
sell open 1 at 71580
```

发生交易后，请在ydClient的账户明细界面中观察报单、成交、持仓和资金的变化情况。

至此，您已经编译并运行了您的第一个易达策略程序，您可以继续尝试其他样例程序并阅读这些程序，有助于帮助您快速掌握API的基础知识。关于ydExample各个例子的说明摘抄如下：

```
我们给出一些使用YDApi的例子。这些例子都使用下面这个针对单一品种的策略：
   如果行情中的买量比卖量大100就按照对手价买入1手
   如果行情中的卖量比买量大100就按照对手价卖出1手
   风控限制是持仓量不允许超过3

下面这些例子给出不同精细程度的持仓管理方式
```

```
Example1:只管理成交持仓，不管理报单持仓，同时只允许至多一张挂单
Example2:基于报单回报管理报单持仓，进行更加精确的持仓管理
Example3:在上个例子的基础上，又增加了基于自身发出的报单来管理报单持仓，以便实现更加精确的持仓管理。此外，该例子也展示了利用notifyCaughtUp获取追上最新流水的信息
Example4:使用YDExtendedApi来实现相同的功能，程序可以大大简化

下面的例子用于展现其他功能
Example5:展现发送期权执行、期权放弃执行和询价指令
Example6:展现一个做市商报价系统如何使用YDExtendedApi
Example7:展现如何使用YDExtendedApi进行简单的风险监控和报警
Example8:展现如何使用YDExtendedApi进行自动建立组合持仓
```

通过学习样例程序，您应该已经掌握了编写易达策略程序的基本步骤，接下来，我们建议您系统性的阅读剩下的内容，我们不仅仅详细说明了易达的基本原理和使用方法，还提供了编写易达API程序的技巧和最佳实践。

如您在阅读过程中遇到任何问题，欢迎直接通过邮件（support@hanlinit.com）或者通过您的经纪商与我们直接取得联系，我们很乐意回答您的任何问题。

2. 基本概念

2.1. API选型

易达提供了ydApi、ydExtendedApi、裸协议、ydCTP以及即将发布的python接口共五种不同类型的交易接口，以满足广大投资者的丰富需求。

裸协议是自由度最高的报单方式，易达向全市场公布了报单和撤单上行协议以及报单回报、成交回报、错单回报等下行协议，投资者可以自行构建UDP报文报撤单或者监听解析下行回报报文，使用时需要配合ydApi或者ydExtendedApi。适合习惯使用裸协议接入柜台的投资者，详情请参考裸协议。

ydCTP是仿CTP的API，模拟了CTP的报单、查询等核心部分功能，使得原先基于CTP开发的程序不经修改即可连接易达柜台，便于尚未正式使用易达的投资者评估、测试易达交易系统的性能和稳定性。由于易达和CTP的体系结构并不兼容，ydCTP无法覆盖CTP的所有接口，也无法做到和原生接口行为完全一致，不建议在生产环境中正式使用ydCTP交易。详情请参考相关文档。

易达将即将发布python版本的易达原生API，如有需要请联系易达客服。

2.1.1. ydApi

ydApi是高性能API，以极简高效为设计理念。负责把投资者的报单送到柜台以及接收到回报后通过回调函数通知投资者，通常情况下API内部不会留存除了静态数据以外的报单、成交等动态数据，更不会帮助投资者计算资金和持仓，因此内存占用极小。使用ydApi的投资者需要自行根据日初持仓以及后续的报单、成交等回报来管理资金和持仓以及提供相应的查询方法。适合高度关注性能且有自行管钱管仓的能力的投资者。

若开启了高可用特性，ydApi会占用与回报数据量大致相当的内存用于高可用恢复。经估测，在Linux64位系统中500万报单无成交的情况下ydApi约占400M内存，实际生产中由于存在成交、错单、组合等，在相同报单量的情况下会占用更多。

ydApi拥有以下特性：

- 线程安全：即在任意线程可以随时调用任何API方法
- 可重入：即在任意回调函数中可以调用任何API方法
- 指针稳定：即ydApi中通过get或者find方法的静态数据的指针在整个生命周期内都是不会改变的，策略程序可以随时读取指针指向的数据；但通过notify回调接口返回的报单回报、成交回报等指针是不同的，OrderSysID相同的多次回报的指针也是不同的。特例是通过notifyCombPosition返回的静态数据的指针是稳定的。

易达目前提供Linux64位、Win32位和Win64位三种版本的API。Linux64位版本的API针对Linux平台做了大量优化，其性能大大优于Windows的两个版本，是易达推荐在生产中使用的版本。对于必须要使用Windows平台的投资者，由于Win32位程序只能使用2G的用户内存，为防止超出内存限制导致API卡死的情况发生，我们建议使用 64位版本。

2.1.2. ydExtendedApi

ydExtendedApi是全能型API，以功能完善和使用方便为设计理念。在继承了ydApi所有功能和特性的基础上（ydExtendedApi是ydApi的子类），API内部保存了所有报单、成交等动态数据，基于这些动态数据帮助投资者管理资金和持仓，并提供了各种丰富的特有数据结构和查询方法，代价是微小的下行性能开销和一定空间占用，当交易量大时占用的空间较大。适合习惯使用全功能API的投资者。

由于ydExtendedApi在本地维护了扩展数据结构，占用的空间相比ydApi较大。经估测，在Linux64位系统中500万报单无成交的情况下，未启用高可用特性时ydExtendedApi约占1400M内存；启动高可用特性时ydExtendedApi约占1800M内存，实际生产中由于存在成交、错单、组合等，在相同报单量的情况下会占用更多。

ydExtendedApi的查询方法都在本地执行，不会到柜台查询，但是由于所有查询都会加锁而导致成本较高，策略程序可以保存从API返回的指针，并在后续执行过程中直接使用指针指向的数据。除了继承了ydApi的指针稳定特性之外，凡是通过ydExtendedApi的get或者find方法获取的，以及YDExtendedListener回调返回的扩展动态数据的指针也都是稳定的。

2.2. 报单方式

易达支持TCP、UDP和XTCP三种报单方式，这些报单方式在穿透性能和可靠性方面有所区别，投资者可以根据自身需求选择报单方式。

2.2.1. TCP报单

TCP报单整体穿透延时比较高，API报单自身的耗时（调用报单起至报单第一个字节出现在光纤上的耗时）约为1-2微秒左右，柜台穿透在8-10微秒左右，但能保证报单必定送达柜台。投资者在测试或者通过互联网连接到柜台时，建议采用TCP的方式，可以避免因为防火墙等原因导致的发单丢失问题。

在TCP报单相同的线程内，还负责发送非交易信息（如修改密码）和接收柜台发送的回报信息，因此无论采用何种报单方式，TCP报单所在的线程一定创建。

要使用TCP报单，请在API配置文件中设置TradingProtocol=TCP。详情请参见配置文件。

2.2.2. UDP报单

UDP报单整体穿透延时低，API报单自身的耗时约为200纳秒左右，柜台穿透约2-3微秒左右（视不同交易所穿透有所区别），易达官方的参考穿透值都是在UDP报单方式下测得的，我们建议投资者在生产环境中使用UDP报单。UDP报单仅用于报送上行报单，回报仍然通过原TCP通道返回。

对于部分投资者担心UDP可能会丢单的问题，首先，易达柜台所在的网络条件通常是稳定、快速和空闲的，除非模块、光纤等发生故障，否则极少会发生UDP丢失的情况；其次，易达提供了柜台回报功能，投资者收到柜台回报即代表柜台已经收到了报单，这样即便丢失了投资者也能知道，详情请参见报单回报中关于柜台回报的功能描述。

要使用UDP报单，请在API配置文件中设置TradingProtocol=UDP。详情请参见配置文件。若柜台没有开启UDP服务，那么每次调用报单接口都将返回false。

2.2.3. XTCP报单

XTCP报单整体穿透延时较低，API报单自身的耗时约为250纳秒左右，柜台穿透相比UDP报单增加100ns以内的延时。其运行模式与UDP相同，仅用于报送上行报单，回报仍然通过原TCP通道返回。

相比UDP报单，XTCP完全消除了丢单的问题且增加穿透时间较小，适合于不信任UDP协议的投资者使用。但由于TCP连接协议栈维护的开销相比UDP较大，因此建议投资者谨慎使用XTCP连接，若大范围使用XTCP线程，将会对策略机和柜台造成较大压力。

要使用XTCP报单，请在API配置文件中设置TradingProtocol=XTCP。详情请参见配置文件。若柜台没有开启XTCP服务，那么将会降级使用TCP报单。由于XTCP服务默认是关闭的，请在使用前与经纪商确认是否打开了XTCP服务。

2.3. API线程

易达API启动完成后，会创建TCP回报接收、TCP行情接收和定时器三个线程；报单是通过调用insertOrder的线程直接发出的，并没有专门的线程处理报单发送。

2.3.1. TCP回报接收线程

TCP回报接收线程的主要工作是接收回报并回调YDListener以及向柜台发送心跳，YDListener或者YDExtendedListener的回调基本（notifyMarketData除外）都是从本线程发起，若长时间（60秒）停留在回调函数中不退出会引起心跳超时而导致本线程的TCP连接断线，因此建议投资者在每个回调函数的处理时间尽可能短。可以在API配置文件中设置TCPTradingCPUID指定需要绑定CPU核心。

TCP回报接收线程有忙查询和select两种网络监听模式。若采用忙查询模式，则收取回报的速度较快，但是其所在CPU的使用率会达到100%；若采用select模式，则会根据设定时间等待select超时后继续下一次select，对CPU几乎没有开销，但是收取回报的速度较忙查询慢，可以在API配置文件中设置TradingServerTimieout设置超时时间，具体请参考交易配置。

为保证柜台上所有连接接收流水的公平性，柜台每向一个连接推送20条流水，就会切换到下一个连接，这样就可以避免某账户盘中登陆后收取全量流水时对其他已有连接接收回报产生影响。

2.3.2. TCP行情接收线程

TCP行情接收线程的主要工作是接收交易所前置推送的行情并回调YDListener以及向柜台发送心跳，YDListener中的notifyMarketData的回调是从本线程发起的，与TCP回报线程相同的原因，notifyMarketData的处理也需要快速，否则会导致本线程的TCP连接断线。可以在API配置文件中设置TCPMarketDataCPUID指定需要绑定CPU核心。

TCP行情接收线程有忙查询和select两种网络监听模式。若采用忙查询模式，则收取行情的速度较快，但是其所在CPU的使用率会达到100%；若采用select模式，则会根据设定时间等待select超时后继续下一次select，对CPU几乎没有开销，但是收取行情的速度较忙查询慢几微秒，可以在API配置文件中设置TCPMarketDataTimeout设置超时时间，具体请参考行情配置。

2.3.3. 定时器线程

定时器线程是统一执行API内各个定时任务的线程，该线程每隔1毫秒定时唤醒。每次定时器触发都将询问各个定时任务是否执行，由各个定时任务自身的特性决定是否执行。目前定时器线程仅用于资金刷新机制中自动模式通知投资者合适的刷新时机。

定时器线程不可关闭，可以在API配置文件中设置TimerCPUID绑定该线程到指定CPU。

2.4. 版本规则

易达采用四段式版本号编码规则，格式为a.b.c.d，每个发布版本只会修改四段版本号中的一个，我们承诺如下：

- a：协议版本号，改变本段版本号的版本必定会修改协议并导致之前发布的API版本全部不兼容，也可以包含b/c段的修改内容，只有当API和柜台的本段版本号一致且API的a.b.c版本号低于柜台的a.b.c版本号时，API才能在该版本的柜台上正常工作。本段版本号较大的版本是高版本，若相同则继续比较b段。
- b：功能版本号，增加本段版本号的版本必定会增加新功能，也可以包含c段的修改内容。本段版本号较大的版本是高版本，若相同则继续比较c段。
- c：补丁版本号，增加本段版本号的版本只能包含修复Bug的补丁内容，且必须尚未发布的版本。本段版本号较大的版本是高版本，若相同则继续比较d段。
- d：紧急补丁版本号，增加本段版本号的版本**原则上**只能包含修复紧急Bug的补丁内容，且必须是已正式发布的生产版本，但是为了应对快速变化的业务需求，易达可能会不得不在紧急补丁中增加新的功能，但是会保证版本兼容性。本段版本号较大的版本是高版本。

易达承诺所有投资者使用的易达的版本是相同的，不存在所谓的特殊版本。但是按照易达的升级策略和规范，在试运行和升级过程中，部分投资者可能会使用到尚未向全市场正式发布的高版本柜台，请各位投资者谅解：

- 在完成版本内部测试后，易达会邀请个别愿意并有能力承担风险，且在业务和技术方面均拥有较强运维和应急处置能力的经纪商试运行新版本，试运行时间并不固定，版本发布的内容越多试运行时间通常会越长，而且一旦在试运行阶段修复了新的问题或引入了新的功能，试运行时间会进一步延长
- 在试运行通过后，易达会向全市场发布正式版本，此时所有期货公司都可以分批次逐步升级到最新版本。为了保证易达有充足的服务人力应对升级过程中的各种问题，我们在开始的1-2周会限制每周的总升级套数。以易达柜台目前的市场保有量以及部分经纪商采取了较为保守的升级策略，全市场完成升级需要2-3个月时间。

2.4.1. API版本兼容性

只有当API和柜台的a段版本号相同，且API的a.b.c段版本号介于MinApiVersion和MaxApiVersion之间，API是才能正常连接和登录柜台，否则会收到YD_ERROR_TooHighApiVersion=62或者YD_ERROR_TooLowApiVersion=58的登陆报错信息。

- 若经纪商未做特别设定，柜台默认的MaxApiVersion等于柜台自身的版本。在特殊情况下，可以通过配置柜台参数MaxApiVersion修改柜台支持的最高API版本，MaxApiVersion只能设置a.b.c段版本

号。只比较a.b.c段版本号是为了能让API的生产紧急补丁（修改版本号第四位）适配低版本的柜台。

- 若经纪商未做特别设定，柜台默认的MinApiVersion为1.0.0。

柜台当前版本、MaxApiVersion和MinApiVersion的获取方式参见系统参数。

API的版本可以通过以下两种不同的方法获取，也可以在log日志中查看API版本号，详情参见写日志：

```
class YDApi
{
  virtual const char *getVersion(void)=0;
}

/// Same as getVersion inside YDApi, put here to get version without make api
YD_API_EXPORT const char *getYDVersion(void);
```

3. 生命周期

3.1. 创建

要创建API实例，首先需要确定使用ydApi还是ydExtendedApi，这两种API的概要差异可以参考API选型，详细差异请查阅本文相关内容。

3.1.1. 创建ydApi

若选择使用ydApi，那么创建过程大致如下：

```
// 创建YDApi
YDApi *pApi=makeYDApi(configFilename);
if (pApi==NULL)
{
  printf("can not create API\n");
  exit(1);
}

// 创建Api的监听器
YDExampleListener *pListener=new YDExampleListener(...);
/// 启动Api
if (!pApi->start(pListener))
{
  printf("can not start API\n");
  exit(1);
}
```

makeYDApi需要传入一个配置文件，该配置文件的配置方法和样例参见配置文件。

上面例子中的YDExampleListener是投资者自行编写的YDListener的子类，所有回报类信息都通过该类的回调函数通知到策略程序，由于YDListener的实例是由策略程序创建的，策略程序应该维护其生命周期，在需要的时候自行销毁。YDListener中的回调函数将分布在本文各个功能章节中分别介绍。

调用ydApi.start即可启动API，参数pListener不可以为空，该函数调用后会立即返回并不阻塞。为防止程序直接退出，策略程序需要调用成功后增加阻塞代码。

```
virtual bool start(YDListener *pListener)=0;
```

3.1.2. 创建ydExtendedApi

若选择使用ydExtendedApi，那么创建过程大致如下：

```
// 创建YDApi
YDExtendedApi *pApi=makeYDExtendedApi(configFilename);
if (pApi==NULL)
```

```
{
  printf("can not create API\n");
  exit(1);
}
// 创建Api的监听器
YDExampleListener *pListener=new YDExampleListener(...);
/// 启动Api
if (!pApi->start(pListener))
{
  printf("can not start API\n");
  exit(1);
}
```

可以看到，除了调用makeYDExtendedApi创建API实例外，其余部分与ydApi完全相同。

为方便ydExtendedApi的用户获得扩展消息的变动通知，在启动API时可以使用startExtended同时接收YDListener和YDExtendedListener的回调通知。

```
// 创建YDApi
YDExtendedApi *pApi=makeYDExtendedApi(configFilename);
if (pApi==NULL)
{
  printf("can not create API\n");
  exit(1);
}
// 创建Api的监听器
YDExample7Listener *pListener=new YDExample7Listener(...);
/// 启动Api，这里使用了YDExtendedListener
if (!pApi->startExtended(pListener,pListener))
{
  printf("can not start API\n");
  exit(1);
}
```

startExtended的函数签名如下，参数pListener和pExtendedListener均不可以为空，除了增加了YDExtendedListener回调通知之外，本函数与start没有任何区别：

```
virtual bool startExtended(YDListener *pListener,YDExtendedListener
*pExtendedListener)=0;
```

YDExtendedListener的定义如下，与YDListener返回的结构体相比，YDExtendedListener返回的扩展结构体包含了更多信息，可以方便投资者编写代码。各个扩展结构体的具体内容请参见ydDataStruct.h。

```
class YDExtendedListener
{
public:
```

```
  virtual ~YDExtendedListener(void)
  {
  }
  // all address of parameters in following methods are fixed
  virtual void notifyExtendedOrder(const YDExtendedOrder *pOrder)
  {
  }
  virtual void notifyExtendedTrade(const YDExtendedTrade *pTrade)
  {
  }
  virtual void notifyExtendedQuote(const YDExtendedQuote *pQuote)
  {
  }
  virtual void notifyExtendedPosition(const YDExtendedPosition *pPosition)
  {
  }
  virtual void notifyExtendedAccount(const YDExtendedAccount *pAccount)
  {
  }
  // notifyExchangeCombPositionDetail and notifyExtendedSpotPosition will only be used when trading SSE/SZSE
  virtual void notifyExchangeCombPositionDetail(const YDExtendedCombPositionDetail *pCombPositionDetail)
  {
  }
  virtual void notifyExtendedSpotPosition(const YDExtendedSpotPosition *pSpotPosition)
  {
  }
};
```

3.1.3. 配置文件

在调用makeYDApi方法时需要指定客户端使用的配置文件。配置文件中的参数主要包括三类：

- 网络配置，包括ydServer的IP地址和端口，其中端口请始终填写期货公司提供的TCP端口，其他端口包括UDP报单端口、TCP行情端口会在登录后自动下发，是否启用UDP报单是由UDPTrading参数控制的。
- 交易配置，包括是否使用UDP报单，以及接收回报线程的工作方式。
- 行情配置，包括是否接收ydServer的TCP或者UDP行情。

以下为ydApi提供的用于生产环境的客户端配置文件最佳实践模板，该模板最终实现的效果是：

- 使用UDP报单
- 忙查询接收TCP回报，加快回报的获取速度，并把接收线程绑定在3号CPU
- 打开了资金重算RecalcMode功能，ConnectTCPMarketData=no的设置会被覆盖为yes，即接收行情

```
##########################################
######### Network configurations  #########
##########################################

# Count of recovery site. Used to achieve high availablity at the expense of a little performance of order notification.
# 0 for no recovery, 1 for recovery always use primary site, 2 for recovery use primary and secondary sites
RecoverySiteCount=0

# IP address of primary trading server
TradingServerIP=127.0.0.1

# TCP port of primary trading server, other ports will be delivered after logged in
TradingServerPort=51000

# IP address of secondary trading server.
# Valid only when RecoverySiteCount equals to 2.
TradingServerIP2=

# TCP port of secondary trading server, other ports will be delivered after logged in.
# Valid only when RecoverySiteCount equals to 2.
TradingServerPort2=

##########################################
######### Trading configurations  #########
##########################################

# Choose trading protocol, optional values are TCP, UDP, XTCP
TradingProtocol=UDP

# Affinity CPU ID for thread to receive order/trade notifications, -1 indicate no need to set CPU affinity
TCPTradingCPUID=-1

# Affinity CPU ID for thread to send XTCP trading, -1 indicate no need to set CPU affinity
XTCPTradingCPUID=-1

# Timeout of select() when receiving order/trade notifications, in millisec. -1 indicates running without select()
TradingServerTimeout=-1

# Work mode for recalculation of margin and position profit. Valid when using ydExtendedApi.
```

```
#     auto(default): subscribe market data and automatically recalculate in proper time.
#     subscribeOnly: subscribe market data and recalcMarginAndPositionProfit should be called explicitly
#     off: never do recalculation
RecalcMode=auto

# Gap between recalculations, in milliseconds. Valid when RecalcMode is set to auto.
# It will be adjusted to 1000 if less than 1000
RecalcMarginPositionProfitGap=1000

# Delay of recalculation after market data arrives to avoid collision with input order, in milliseconds.
# Valid when RecalcMode is set to auto. Should be between 0 and 100.
RecalcFreeGap=100

##########################################
####### MarketData configurations  ########
##########################################

# Whether need to connect to TCP market data server
ConnectTCPMarketData=no

# Timeout of select() when receiving TCP market data, in millisec. -1 indicates running without select()
TCPMarketDataTimeout=10

# Affinity CPU ID for thread to receive TCP market data, -1 indicate no need to set CPU affinity
TCPMarketDataCPUID=-1

# Whether need to receive UDP multicast market data
ReceiveUDPMarketData=no

##########################################
######### Other configurations  ###########
##########################################

AppID=yd_dev_1.0
AuthCode=ecbf8f06469eba63956b79705d95a603
```

3.1.3.1. 网络配置

网络配置包含易达柜台的IP地址和端口，生产环境交易时，请向所在期货公司技术部门询问具体配置情况。

RecoverySiteCount：高可用节点个数。当为0时，不启用API高可用特性；当为1时，启用单柜台高可用特性，当柜台重启后，API会重新连接柜台并尽可能恢复到最新状态，从而不会导致API直接退出；当为2时，启用主备双柜台高可用特性，当主柜台因故障停止备用柜台（在TradingServerIP2和TradingServerPort2中指定）启动后，API会重新连接柜台并恢复到最新状态。使用高可用特性，会损失极为微小的回报性能。

TradingServerIP：易达柜台的IP地址

TradingServerPort：易达柜台的端口。易达柜台通常会开放三个端口，分别是TCP交易、TCP行情和UDP交易，XTCP交易和UDP行情端口默认关闭。TradingServerPort需要填入的是TCP交易的端口，其他端口会在登录成功后由柜台自动下发到客户端。请不要将UDP交易端口填入TradingServerPort，否则会导致无法连接易达柜台，如果你希望通过UDP报单，请通过UDPTrading=yes参数控制。、

UDPTradingServerPort：UDP交易的端口。由柜台自动下发，通常不要填写。当需要从外网访问内网柜台而做了NAT时，对外的UDP交易端口可能会发生改变，此时下发的端口将无法正确接收UDP报单数据，需要根据期货公司提供的实际端口配置该参数。

XTCPTradingServerPort：XTCP的交易端口。由柜台自动下发，通常不要填写。当需要从外网访问内网柜台而做了NAT时，对外的XTCP交易端口可能会发生改变，此时下发的端口将无法正确接收XTCP报单数据，需要根据期货公司提供的实际端口配置该参数。

TCPMarketDataServerPort：TCP行情的端口。由柜台自动下发，通常不要填写。当需要从外网访问内网柜台而做了NAT时，对外的行情端口可能会发生改变，此时下发的端口将无法正确接收行情，需要根据期货公司提供的实际端口配置该参数。

TradingServerIP2：易达备用柜台的IP地址。RecoverySiteCount为2时生效。

TradingServerPort2：易达备用柜台的端口。设置方法同TradingServerPort。RecoverySiteCount为2时生效。

UDPTradingServerPort2：易达备用柜台的UDP交易端口，使用方法同UDPTradingServerPort。RecoverySiteCount为2时生效。

XTCPTradingServerPort2：易达备用柜台的XTCP的交易端口，使用方法同XTCPTradingServerPort。RecoverySiteCount为2时生效。

TCPMarketDataServerPort2：易达备用柜台的TCP行情端口，使用方法同TCPMarketDataServerPort。RecoverySiteCount为2时生效。

3.1.3.2. 交易配置

TradingProtocol：报单方式，可选报单方式为TCP、UDP、XTCP。由于UDP和XTCP总体穿透性能相比TCP好很多，我们建议在生产环境使用UDP或XTCP报单。为保证兼容性，若没有设置TradingProtocol，则会使用原有配置UDPTrading。

TCPTradingCPUID：设置TCP发送报单和接收回报线程的亲和性。-1代表不需要设置亲和性；否则为CPU/Core的编号。

XTCPTradingCPUID：设置XTCP发送报单线程的亲和性。-1代表不需要设置亲和性；否则为CPU/Core的编号。

TradingServerTimeout：设置该参数指示YDListener的TCP交易接收线程如何使用监听网络通信。如果设置为正整数值，表示该线程使用POSIX的select机制去监听是否有网络数据到达，监听时间间隔为所指定的值，单位为毫秒。设置为-1，接收线程将以非阻塞方式读取信息，其CPU使用率会达到100%，如果客户端未根据自身机器的配置情况合理分配CPU/Core，设置线程与CPU/Core的亲和性，将严重影响客户端程序的运行性能，延迟会显著增加。

RecalcMode：保证金和持仓盈亏的重算模式。默认值为auto。

- auto：完全由API负责重算保证金和持仓盈亏，API会自动订阅计算需要的行情。可以通过RecalcMarginPositionProfitGap和RecalcFreeGap这两个参数控制刷新间隔和距离行情的延时时间。会强制接收TCP行情
- subscribeOnly：只帮助自动订阅需要的行情，但是需要由投资者自行调用recalcMarginAndPositionProfit重算方法。会强制接收TCP行情
- off：不重算保证金和持仓盈亏。通常投资者不应该使用这个模式

RecalcMarginPositionProfitGap：两次重算之间的间隔，单位为毫秒，默认为1000毫秒。最小值为1000，如果小于1000，系统自动会调整为1000。本参数只有当RecalcMode为auto时才有效。

RecalcFreeGap：距离行情的延时间隔。为了尽可能避免与报单发生冲突，要尽可能避免在行情到来的时候重算，投资者可以根据不同交易所调整参数使得行情在两个时间切片报单较为空闲的时间重算。单位为毫秒，默认值为100毫秒，必须介于0和100之间，否则大于100的值会被调整为100，小于0的值会被调整为0。本参数只有当RecalcMode为auto时才有效。

3.1.3.3. 行情配置

请注意，无论是TCP行情还是UDP组播行情，易达的行情均不是高速行情，目前仅可用于易达服务端和客户端用于计算保证金和盈亏。如在生产交易中需要行情，请联系期货公司接收组播行情。

ConnectTCPMarketData：是否连接ydServer的TCP行情服务。yes表示接收，no表示不接收。

TCPMarketDataTimeout：设置该参数指示YDListener的TCP行情接收线程如何使用监听网络通信。如果设置为正整数值，表示该线程使用POSIX的select机制去监听是否有网络数据到达，监听时间间隔为所指定的值，单位为毫秒。设置为-1，接收线程将以非阻塞方式读取信息，其CPU使用率会达到100%，如果客户端未根据自身机器的配置情况合理分配CPU/Core，设置线程与CPU/Core的亲和性，将严重影响客户端程序的运行性能，延迟会显著增加。

TCPMarketDataCPUID：设置TCP行情接收线程的亲和性。-1代表不需要设置亲和性；否则为CPU/Core的编号。

ReceiveUDPMarketData：是否接收ydServer发送的UDP组播行情。目前所有易达柜台的UDP组播行情功能均是关闭的，请始终保持该参数为no。

3.1.3.4. 其他配置

TimerCPUID：用于设置API中定时器线程的CPUID。

AppID和AuthCode：投资者可以在配置文件中设置AppID和AuthCode，并在登陆调用login时appID和authCode分别填入NULL，API就会使用配置文件中配置的值。

LogDir：指定ydApi产生日志的目录，默认为log。目录不会自动产生，需要手工创建，如果没有对应的目录存在，则不会产生日志文件。

3.1.3.5. 自定义配置

易达支持读取配置文件中的配置内容，投资者即可以读取易达API的配置信息，投资者又可以将自定义的配置放在易达配置文件中，然后通过易达API读取。

当投资者设定自定义配置项时，请以“应用名.参数名”的格式予以命名，例如MyApp.Username，如果直接输入没有应用名的参数，该参数仍然是可以使用的，但是API在启动时会报告该参数没有被使用的警告信息，为了避免误解，我们不建议使用没有应用名分隔的参数。

易达自定义配置支持列表类型的参数，可以设置多个同名参数实现这个效果。设置样例如下所示：

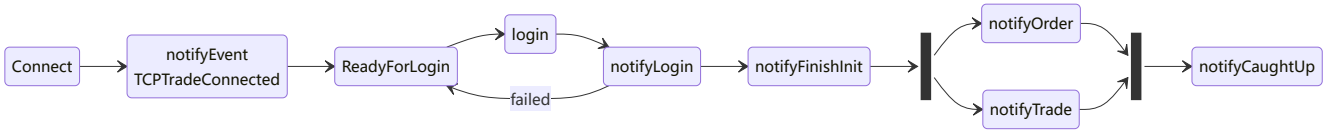
```
MyApp.TradeExchange=SHFE
MyApp.TradeExchange=INE
MyApp.TradeExchange=CFFEX
MyApp.TradeExchange=DCE
MyApp.TradeExchange=CZCE
```

易达API提供了以下两个API获取自定义参数。第一个方法可以获取单个自定义参数，当该参数是列表参数时，获取的是该列表的第一个设置值。第二个方法可以获取列表参数的所有参数值，返回的结果集在用完后需要投资者手动调用destory()销毁，当用于单个自定义参数时也可以返回该参数的值，只不过是放在YDQueryResult集合中的，用完后同样需要调用destory()销毁由API分配的空间。

```
/// get first config using this name in config file (parameter of makeYDApi or makeYDExtendedApi), NULL if not found
virtual const char *getConfig(const char *name)=0;
/// get all configs using this name,  user should call destroy method of return object to free memory after using
virtual YDQueryResult<char> *getConfigs(const char *name)=0;
```

3.2. 启动

在创建了API实例并调用start或者startExtend启动API后，API会按照以下启动流程完成启动初始化工作。



3.2.1. 连接

实例启动后，就会持续连接TradingServerIP和TradingServerPort指定的柜台地址和TCP端口，直到连接成功后，会通过notifyEvent(YD_AE_TCPTradeConnected)回调消息通知已连接成功。

若API配置了双柜台高可用模式，那么会轮询TradingServerIP和TradingServerPort指定的主柜台地址和TCP端口以及TradingServerIP2和TradingServerPort2指定的备柜台地址和TCP端口，直到连接成功后，会通过notifyEvent(YD_AE_TCPTradeConnected)回调消息通知已连接成功。

3.2.2. 登陆

连接成功后，会立即通过回调函数notifyReadyForLogin通知策略程序API已准备好，可以尝试登录。该回调函数的参数hadLoginFailed是指发起本次notifyReadyForLogin回调的原因是否为上次登录失败，若原因是上次登录失败，用户没有必要用完全相同的信息再次登录，因为结果必定失败。

```
virtual void notifyReadyForLogin(bool hasLoginFailed)
```

策略程序在收到notifyReadyForLogin回调通知后，通常应该立即调用login发起登录。login函数的签名如下，登录时必须输入用户名username和密码password，穿透式监管信息appID和authCode是可选的：

- 若appID和authCode设置为NULL，API会使用配置文件中指定的AppID和AuthCode，如果配置文件中没有配置这两个参数则会出错
- 若appID和authCode设置为具体的值，则API只会使用接口中指定的参数而不会使用配置文件中指定的值

```
virtual bool login(const char *username,const char *password,const char
*appID,const char *authCode)=0;
```

下面是比较常见的登陆的代码样例：

```
virtual void notifyReadyForLogin(bool hasLoginFailed)
{
  if (!m_pApi->login(m_username,m_password,NULL,NULL))
  {
    printf("can not login\n");
    exit(1);
  }
}
```

登陆结果会通过notifyLogin回调返回：

```
virtual void notifyLogin(int errorNo,int maxOrderRef,bool isMonitor)
```

若notifyLogin的errorNo为0，则表示登陆成功。isMonitor为true则表示为当前登陆用户为柜台管理员，普通投资者账户登陆则必然为false。maxOrderRef表示在登陆时刻柜台0号报单组OrderGroupID已收到的最大OrderRef，如需获取所有报单组的最大OrderRef请参考报单组。除了以下三种情况外，无论报单、报价单成功与否，只要新报单、报价单所在报单组的OrderRef大于该组当前的MaxOrderRef，都将修改当前账户的MaxOrderRef为该报单的OrderRef：

- 报单、报价单因为报单号单调递增检查而导致的错单不会计入MaxOrderRef
- 报单、报价单因柜台接收到的网络数据包错误而导致无法识别报单账户的错单不会计入MaxOrderRef，通常不应发生本错误，多见于投资者错误破解我司协议后的报单、报价单
- 在不支持报价指令的柜台（没有做市商功能的授权）上报入报价单而导致的错单不会计入MaxOrderRef

登陆成功时，API就能收到柜台下发的TCP行情端口和UDP交易端口，因此，易达不需要在配置文件中指定行情端口、UDP交易端口与XTCP交易端口。但是在某些场景下，如从互联网访问柜台时，通常经过映射后的互联网端口与从柜台下发的端口是不同的，如果不做处理，会表现为TCP行情连不上或者UDP交易无法发送到柜台。在这种情况下，可以手工设置TCP行情端口或者UDP交易端口以覆盖自动下发的端口，可使用TCPMarketDataServerPort覆盖行情端口，UDPTradingServerPort覆盖UDP交易端口，XTCPTradingServerPort覆盖XTCP交易端口；若配置了双柜台高可用模式，可以使用TCPMarketDataServerPort2覆盖备用柜台的行情端口，UDPTradingServerPort2覆盖备用柜台的UDP交易端口，XTCPTradingServerPort2覆盖备用柜台的XTCP交易端口。

若notifyLogin的errorNo非0，则表示登陆失败。errorNo会告知具体失败的原因，此时maxOrderRef和isMonitor都是无意义的，请不要使用。API会等待3秒钟后继续回调notifyReadyForLogin等待用户发送登陆请求，此时notifyReadyForLogin中的hasLoginFailed会被设置为true。登陆失败的原因可能是以下几种：

错误码	错误定义	说明
12	YD_ERROR_InvalidClientAPP	AppID或AppAuthCode错误
18	YD_ERROR_AlreadyLogined	当前API已完成登陆，不可以重复登陆
19	YD_ERROR_PasswordError	登陆密码错误
20	YD_ERROR_TooManyRequests	登录请求数量超过限制
21	YD_ERROR_InvalidUsername	登陆用户不存在
27	YD_ERROR_InvalidAddress	客户端IP地址不在允许登陆的地址段，仅对管理员登陆生效
35	YD_ERROR_ClientReportError	穿透式监管信息采集失败。详情请参见穿透式监管。

错误码	错误定义	说明
50	YD_ERROR_TooManyLogines	易达柜台的授权限制了同时可以登录柜台的账户数，当达到登录上限后，后续连接的账户将无法登陆，必须要等待现有账户退出登陆后，其他账户才能继续登陆。一个账户无论有多少个连接，都只计算一个登陆账户数，换而言之，只有当账户所有的连接都退出后，即YDAccount.LoginCount为0，该账户才能被视为退出。
58	YD_ERROR_TooLowApiVersion	API版本低于柜台要求的最低版本，详情参见API版本兼容性
62	YD_ERROR_TooHighApiVersion	API版本高于柜台支持的最高版本，详情参见API版本兼容性

3.2.2.1. 交易日

交易日会随着登陆回报一起传回到API，从notifyLogin回调开始投资者即可以使用getTradingDay获取交易日。

```
virtual int getTradingDay(void)
```

易达柜台的交易日是根据系统时间计算得出的（日盘时设置为当前自然日的日期，夜盘24:00:00前为下一非节假日的日期，夜盘24:00:00后，若当前自然日为交易日，则为当前自然日，若当前自然日为节假日，则为下一非节假日的日期），生产环境计算得到的交易日通常是准确的，但是在测试环境中很有可能与YDMarketData.TradingDay是不一致的，导致不一致的原因可能是使用了过去的日初数据、交易所的测试环境没有切换交易日或者在周末测试的时候人为制造的交易日。

3.2.2.2. 穿透式监管

根据监管要求，穿透式监管主要分为两个部分，一是登录时需要指定客户端系统的AppID和AuthCode，二是需要上报登陆系统所在服务器的信息。下面我们分别介绍这两个机制。

3.2.2.2.1. AppID和AuthCode

AppID和AuthCode本质上是策略系统等使用API接入柜台的客户端系统的用户名密码，客户端系统要连入柜台首先得满足经纪商的接入测试要求，测试通过后经纪商会将代表该系统的AppID和AuthCode加入到柜台中，而后任何账户都可以使用该客户端系统连入柜台。显然，客户端系统的AppID和AuthCode与投资者账户的用户名和密码并没有绑定关系。

> 易达提供的ydClient客户端有其固化在客户端内部的AppID和AuthCode，该客户端的AppID和AuthCode也被默认添加在柜台的允许系统清单中，因此任何投资者都可以使用ydClient登陆到柜台。

易达并不限制AppID的命名格式，但监管对AppID是有要求的，我们建议按照监管要求命名您的客户端系统。监管对AppID的要求如下：

- AppID由三部分组成，分别是厂商名称、软件名和版本号，各部分最大长度分别为10、10、8个字符
- AppID应遵循以下格式：厂商名称_软件名称_版本号，例如yd_client_1.0
- 对于个人开发的终端软件，厂商名称应固定为client。由于使用易达系统的大部分投资者都是自研系统，因此厂商名称应固定为client

由于AuthCode是属于客户端系统的信息，因此录入易达柜台的AuthCode需要由投资者指定，以确保该客户端系统在所有柜台系统中都有相同的授权码。

易达在检查AppID和AuthCode信息对的时候，只是简单比较字符是否与柜台后台清单中的信息一直，并不会把AuthCode作为加密信息来处理。

3.2.2.2.2. 信息采集

监管需要采集的信息包括两部分，一部分是必须从客户端服务器上采集，一部分可以在柜台端采集。易达API会自动从客户服务器上采集监管要求的客户端服务器信息，不同类型的终端操作系统采集的信息有所区别，下面我们仅列出从客户端服务器上采集的信息：

```
Windows:终端类型(固定为1)@信息采集时间@私网IP1@私网IP2@网卡MAC地址1@网卡MAC地址2@设备名@操作系统版本@硬盘序列号@CPU序列号@BIOS序列号@系统盘分区信息
Linux:终端类型(固定为2)@信息采集时间@私网IP1@私网IP2@网卡MAC地址1@网卡MAC地址2@设备名@操作系统版本@硬盘序列号@CPU序列号@BIOS序列号
```

以下是两种操作系统采集到的信息的样例，如果投资者想要查看API采集和上报到柜台的信息（API版本必须高于1.122），可以在程序当前目录下创建log目录，就能在完成登陆后在log目录下的日志中看到如下所示的穿透式采集信息：

```
1@2022-08-12 09:13:13@192.168.6.1@192.168.80.1@005056C00001@005056C00008@MI@6.2@0100_0000_0000_0@BFEBFBFF000806C1@33481/01QV@C,FA668A82,NTFS,475
2@2022-08-10 20:50:02@192.168.15.21@@5254003A3DF5@@test@3.10.@@078BFBFF00050654@Not Specified
```

> 在Linux系统的采集过程中，我们会使用dmidecode程序采集BIOS序列号，通常若不做特殊设置普通用户是没有该程序的执行权限的，运行客户端系统的用户必须具有该程序的执行权限才能确保信息采集完全。易达会首先检查dmidecode是否被赋予了suid，若被赋予则直接执行，若没有被赋予，则使用sudo dmidecode的方式执行。投资者可以通过chmod +x /usr/sbin/dmidecode命令为该程序添加suid。

当柜台参数"客户端报告检查方式ClientReportCheck"被设置为强制enforce时，若柜台检查采集信息发现采集不完整时，例如上面第二条Linux样例中的硬盘序列号未采集到，会禁止客户端登陆，并产生YD_ERROR_ClientReportError的登陆错误。下面列出了易达API采集信息采用的操作系统命令，若发现采集信息有缺失的，请手工执行以对应的采集方法确认命令执行结果是否正常，请使用与启动客户端系统相同操作系统用户执行命令，确保能发现权限不足导致的问题（大部分取不到数据的原因）：

- Windwos的硬盘序列号：Get-WmiObject -Query “SELECT SerialNumber FROM Win32_PhysicalMedia”
- Windows的CPU序列号：Get-WmiObject -Query “SELECT ProcessorID FROM Win32_Processor”
- Windows的BIOS序列号：Get-WmiObject -Query “SELECT SerialNumber FROM Win32_BIOS”
- Linux的硬盘序列号：首先通过/bin/lsblk -dn -o TYPE,NAME找到TYPE为disk的设备名NAME，然后调用/sbin/udevadm info –query=all –name=/dev/{NAME}获得该设备的序列号
- Linux的BIOS序列号：/usr/sbin/dmidecode -s system-serial-number

3.2.3. 收取静态数据

为减少柜台查询对柜台产生的冲击，易达所有的查询都是在本地执行的，这样就要求客户端和服务端有完全相同的数据，这一原则对ydApi和ydExtendedApi均有效，区别在于ydApi里面只有静态数据，而ydExtendedApi除了同样拥有静态数据之外，还与柜台端一样管理了资金、持仓、报单和成交等动态数据，因此，在ydExtendedApi上查询资金、持仓、报单和成交等也是在本地执行的。

静态数据是指在一个交易日内不会发生改变的数据，如合约、日初保证金率、日初手续费率、账户设置等，每一种日初数据都会有其对应的结构体存放，例如合约数据对应YDInstrument、账户数据对应YDAccount。由于某些结构体同时存在静态数据和动态数据，例如行情结构体YDMarketData中的昨结算价PreSettlementPrice是静态数据，但最新价LastPrice是动态数据，而在收取静态数据阶段，易达柜台只会下发静态数据而不保证结构体中动态数据部分的正确性，因此，在收取静态数据阶段务必只使用各结构体中的静态数据而不要使用动态数据，否则可能会发生未知的错误。各个数据结构体中的静态和动态部分会在下文分别指出。如未特别说明，则对应于这些静态数据的整个结构体内的所有字段都是静态的。

登陆成功以后，柜台就开始推送静态数据，API接收完成后即建立行情连接和XTCP连接（如果启用的话），无论是否连接成功，都会回调notifyFinishInit以通知策略程序所有静态数据已加载完成，这给予投资者一个时间点来为后续接收报单、成交回报做好准备，包括获取日初资金、持仓、保证金率、费率等信息。

```
1 virtual void notifyFinishInit(void)
```

除了通过回调函数知道静态数据推送完成外，易达API还提供了同步查询函数来查询静态数据是否推送完成，如果不希望在静态函数接收阶段收到回调消息，那么可以使用本函数判断当前状态过滤。

```
1 virtual bool hasFinishedInit(void)=0
```

下面我们将逐一介绍易达的静态数据，以下内容中涉及到的函数在notifyFinishedInit回调中以及之后就可以正常使用。

3.2.3.1. 交易所

易达目前支持中金所、上期所、能源所、大商所、广期所、郑商所、上交所（期权）、深交所（期权）共8个交易所，也支持在一套易达柜台上同时配置所有交易所连接，常见的是上期所和能源所配置在一套易达柜台上，因此，易达提供了遍历和查找交易所的两组方法，如下所示：

```
virtual int getExchangeCount(void)=0
virtual const YDExchange *getExchange(int pos)=0
virtual const YDExchange *getExchangeByID(const char *exchangeID)=0
```

前两个方法可以遍历所有交易所，具体方法如下面代码所示：

```
for(int i = 0; i < pApi->getExchangeCount(); i++) {
  YDExchange *exchange = pApi->getExchange(i);
}
```

最后一个方法可以指定查找一个交易所，查找各个交易所的调用如下所示：

```
pApi->getExchangeByID("CFFEX")  //中金所
pApi->getExchangeByID("SHFE")   //上期所
pApi->getExchangeByID("INE")    //能源所
pApi->getExchangeByID("DCE")    //大商所
pApi->getExchangeByID("GFEX")   //广期所
pApi->getExchangeByID("CZCE")   //郑商所
pApi->getExchangeByID("SSE")    //上交所
pApi->getExchangeByID("SZSE")   //深交所
```

YDExchange结构体中所有字段均为静态数据，大部分用于表示是否支持某功能，因此，如需详细了解具体某一字段的含义和使用方法，请在本文档中搜索该字段，以获取与该功能有关的完整介绍。

3.2.3.2. 产品

易达提供了遍历和查找产品的两组方法。

```
virtual int getProductCount(void)=0;
virtual const YDProduct *getProduct(int pos)=0;
virtual const YDProduct *getProductByID(const char *productID)=0;
```

前两个方法可以遍历所有产品，若此时易达柜台上配置了多个交易所，则会遍历所有交易所的所有产品，具体方法如下面代码所示：

```
for(int i = 0; i < pApi->getProductCount(); i++) {
  YDProduct *product = pApi->getProduct(i);
}
```

最后一个方法可以指定查找一个产品，查找产品的调用示例如下所示，如需知道易达所有产品的表达式，则请使用上述方法遍历后逐个查看YDProduct.ProductID：

```
pApi->getProductByID("cu")
pApi->getProductByID("cu_o")
pApi->getProductByID("IC")
```

YDProduct结构体中所有字段均为静态数据，大部分字段与YDInstrument是重复的，其中也包含了原本应该属于合约的属性，例如Multiple、Tick、UnderlyingMultiply、MaxMarketOrderVolume、MinMarketOrderVolume、MaxLimitOrderVolume、MinLimitOrderVolume，其原意是当合约上相应字段没有赋值时可以作为其默认值使用，但是目前所有合约都有妥善设置其属性值，并不会出现空值的情况，因此YDProduct上的合约属性就没有太大意义。如需了解字段含义请参考合约章节相关内容。

m_pMarginProduct用于计算大边保证金和跨产品大边保证金，详情请参见保证金优惠。

3.2.3.3. 合约

易达提供了遍历和查找合约的两组方法。

```
virtual int getInstrumentCount(void)=0
virtual const YDInstrument *getInstrument(int pos)=0
virtual const YDInstrument *getInstrumentByID(const char *instrumentID)=0
```

前两个方法可以遍历所有合约，若此时易达柜台上配置了多个交易所，则会遍历所有交易所的所有合约，具体方法如下面代码所示：

```
for(int i = 0; i < pApi->getInstrumentCount(); i++) {
  YDInstrument *instrument = pApi->getInstrument(i);
}
```

最后一个方法可以指定查找一个合约，查找合约的调用示例如下所示，如需知道易达所有合约的表达式，则请使用上述方法遍历后逐个查看YDInstrument.InstrumentID：

```
pApi->getInstrumentByID("cu2208")
```

YDInstrument是系统的核心数据结构，除了AutoSubscribed和UserSubscribed外其他都是静态数据，下面将介绍其关键部分字段含义：

字段	说明
InstrumentID	合约代码，如cu2208，SP c2211&c2301
InstrumentHint	合约提示信息，适用于上交所/深交所ETF期权，示例：510050C2009M02350
ProductClass	合约所属产品类型 YD_PC_Futures=1：期货 YD_PC_Options=2：期权 YD_PC_Combination=3：组合，如SP c2211&c2301，目前支持大商所和郑商所的套利合约 YD_PC_Index=4：指数，如沪深300指数 YD_PC_Cash=5：现货，如沪深300ETF
DeliveryYear	交割年份，例如2022的整数

字段	说明
DeliveryMonth	交割月份，例如1代表1月份，12代表12月份
ExpireDate	合约到期日，例如20220808的整数
ExpireTradingDayCount	当前交易日距离合约到期日的交易日天数（不含周末和假期），由于该剩余天数的计算依赖于交易日历，因此对于到期日在明年的合约，只有当年底交易所正式发布了明年的交易日历以后才能计算正确，否则计算所需的交易日历是易达自行推算的，可能与根据正式交易日历计算出来的剩余天数有差异 如果是最后交易日，这个值是0
Multiple	合约乘数，相对于底层资产的倍率
Tick	最小变动价位
MaxMarketOrderVolume	市价报单的最大报单量
MinMarketOrderVolume	市价报单的最小报单量，且报单量必须是其倍数
MaxLimitOrderVolume	限价报单的最大报单量
MinLimitOrderVolume	限价报单的最小报单量，且报单量必须是其倍数
OptionsType	期权类型 YD_OT_NotOption=0：非期权 YD_OT_CallOption=1：看涨期权 YD_OT_PutOption=2：看跌期权
StrikePrice	行权价格，只有当合约是期权时有效
m_pUnderlyingInstrument	期权合约的基础合约的指针，可以以此获取基础合约的相关信息，只有当合约是期权时有效
UnderlyingMultiply	基础乘数，期权相对于其基础合约的倍率，只有当合约是期权时有效 对于其他非期权合约始终为1
m_pMarketData	指向行情结构体的指针，可以以此获取行情，这个行情结构体会随着行情持续刷新
m_pLegInstrument[2]	只适用于组合合约。m_pLegInstrument[0]指向组合合约中左腿合约，m_pLegInstrument[1]指向组合合约中右腿合约
LegDirction[2]	只适用于组合合约。LegDirction[0]表示组合合约中左腿合约的买卖方向，LegDirction[1]表示组合合约中右腿合约的买卖方向
m_pCombPositionDef[2][YD_MaxHedgeFlag]	只适用于组合合约。m_pCombPositionDef[0]指向买组合合约时各投保标志的组合定义数组，m_pCombPositionDef[1]指向买组合合约时各投保标志的组合定义数组。投标标志需要减1转化为数组下标

假设某期货的合约乘数Multiply为5，表示一手期货合约对应5单位的底层资产。若一手该期货的期权对应于期货的两手，则该期权的UnderlyingMultiply为2，那么该期权的合约乘数Multiply为$5 \times 2 = 10$，表示一手期权对应10单位的底层资产。

3.2.3.4. 组合持仓定义

大商所、广期所、郑商所、上交所、深交所的组合持仓定义可以通过以下两种API查询。

第一种方法为遍历所有的组合持仓定义，通过getCombPositionDefCount获取所有组合持仓定义的个数n，然后使用getCombPositionDef从0到n-1逐一遍历所有的组合持仓定义。

```
// 通过序号遍历所有组合持仓定义
virtual int getCombPositionDefCount(void)=0;
virtual const YDCombPositionDef *getCombPositionDef(int pos)=0;
```

第二种方法为通过组合持仓定义名称和组合投保标志查询某个组合持仓定义。组合持仓定义名称combPositionID与交易所的定义并不完全一致，请参考下文对组合持仓定义名称和组合投保标志的定义。

```
// 通过组合持仓定义名称和投保标志查询某个组合持仓定义
virtual const YDCombPositionDef *getCombPositionDefByID(const char
*combPositionID,int combHedgeFlag)=0;
```

通过上述API获取的YDCombPositionDef的信息如下所示：

声明	说明
YDInstrumentID CombPositionID	组合持仓定义名称，举例如下： pg2302,-eg2303 c2207-C-2240,-c2207-C-2240 20008571,-20008572
int CombPositionRef	组合持仓定义内部序号
int ExchangeRef	交易所内部序号
int Priority	组合持仓定义优先级，数字越小优先级越高。 可能是-1，表示该交易所的组合持仓不使用优先级管理，例如郑商所
short CombHedgeFlag	组合持仓定义投保标志，可以是以下类型： YD_CHF_SpecSpec：投机-投机 YD_CHF_SpecHedge：投机-套保 YD_CHF_HedgeHedge：套保-套保 YD_CHF_HedgeSpec：套保-投机
short CombPositionType	组合持仓定义类型。各个交易所的组合持仓定义类型如下表所示。

声明	说明
double Parameter	参数，不同交易所的含义不同。 目前仅用于大商所和广期所的期权对锁、买入期权垂直套利、买入期权期货组合，表示组合保证金折扣，0.2表示组合后的保证金是原两腿保证金总和的20%
const YDExchange *m_pExchange	所属YDExchange的指针
const YDInstrument *m_pInstrument[2]	组合持仓定义两腿合约。 两条腿的次序不一定与CombPositionID的次序相同，易达会按照业务处理的必要自行排序
int PositionDirection[2]	对应于m_pInstrument中的两腿的持仓方向
int HedgeFlag[2]	对应于m_pInstrument中的两腿的投保标志
int PositionDate[2]	对应于m_pInstrument中的两腿的持仓日期，目前全部为历史仓

各个交易所的组合持仓类型如下表所示。

交易所	组合持仓类型	说明
大商所	YD_CPT_DCE_FuturesOffset=0	期货对锁
大商所	YD_CPT_DCE_OptionsOffset=1	期权对锁
大商所	YD_CPT_DCE_FuturesCalendarSpread=2	期货跨期
大商所	YD_CPT_DCE_FuturesProductSpread=3	期货跨品种
大商所	YD_CPT_DCE_BuyOptionsVerticalSpread=4	买入期权垂直套利
大商所	YD_CPT_DCE_SellOptionsVerticalSpread=5	卖出期权垂直套利
大商所	YD_CPT_DCE_OptionsStraddle=7	卖出期权跨式
大商所	YD_CPT_DCE_OptionsStrangle=8	卖出期权宽跨式
大商所	YD_CPT_DCE_BuyOptionsCovered=9	买入期权期货组合
大商所	YD_CPT_DCE_SellOptionsCovered=10	卖出期权期货组合
广期所	YD_CPT_GFEX_FuturesOffset=0	期货对锁
广期所	YD_CPT_GFEX_OptionsOffset=1	期权对锁
广期所	YD_CPT_GFEX_FuturesCalendarSpread=2	期货跨期
广期所	YD_CPT_GFEX_FuturesProductSpread=3	期货跨品种

交易所	组合持仓类型	说明
广期所	YD_CPT_GFEX_BuyOptionsVerticalSpread=4	买入期权垂直套利
广期所	YD_CPT_GFEX_SellOptionsVerticalSpread=5	卖出期权垂直套利
广期所	YD_CPT_GFEX_OptionsStraddle=7	卖出期权跨式
广期所	YD_CPT_GFEX_OptionsStrangle=8	卖出期权宽跨式
广期所	YD_CPT_GFEX_BuyOptionsCovered=9	买入期权期货组合
广期所	YD_CPT_GFEX_SellOptionsCovered=10	卖出期权期货组合
郑商所	YD_CPT_CZCE_Spread=50	套利
郑商所	YD_CPT_CZCE_StraddleStrangle=51	卖出跨式或宽跨式
郑商所	YD_CPT_CZCE_SellOptionConvered=52	卖出期权期货组合
上交所、深交所	YD_CPT_StockOption_CNSJC=100	认购牛市价差
上交所、深交所	YD_CPT_StockOption_CXSJC=101	认购熊市价差
上交所、深交所	YD_CPT_StockOption_PNSJC=102	认沽牛市价差
上交所、深交所	YD_CPT_StockOption_PXSJC=103	认沽熊市价差
上交所、深交所	YD_CPT_StockOption_KS=104	跨式空头
上交所、深交所	YD_CPT_StockOption_KKS=105	宽跨式空头

组合持仓定义的数量非常庞大，特别是大商所目前已经达到了17万条之巨，这将直接导致下发组合持仓定义的耗时过长。为了解决这个问题，在1.108版本的柜台支持以压缩方式发出组合持仓定义，启用条件是设置柜台的MinApiVersion高于1.108即可，这就要求该柜台上所有投资者的API版本都高于1.108。

3.2.3.5. 日初行情

要获取日初行情，需要通过合约的m_pMarketData指针访问行情结构体YDMarketData。

YDMarketData中大部分都是动态数据，只有与上一交易日有关的字段是静态数据，静态数据部分如下表所示：

字段	说明
TradingDay	当前交易日
PreSettlementPrice	昨结算价
PreClosePrice	昨收盘价
PreOpenInterest	昨持仓量

字段	说明
UpperLimitPrice	涨停板价
LowerLimitPrice	跌停板价

3.2.3.6. 账户

对于普通投资者，易达系统只提供了一种获取账户的方式，该方法只能由投资者使用，管理员用户不可以使用该方法。

```
// ydApi和ydExtendedApi均可调用
virtual const YDAccount *getMyAccount(void)

// 只有ydExtendedApi可调用
virtual const YDExtendedAccount *getExtendedAccount(const YDAccount *pAccount=NULL)
```

YDAccount和YDExtendedAccount中的静态数据如下表所示：

字段	说明
AccountID	资金账户
PreBalance	昨余额
WarningLevel1	风险度报警级别1。已无实际作用，请忽略
WarningLevel2	风险度报警级别2。已无实际作用，请忽略
AccountFlag	账户功能开关

AccountFlag是一个位图，每一个二进制位置1表示启用该功能，置0表示关闭该功能。假设投资者开通了：柜台回报、平仓缓冻结和检查报单号三个功能，那么他的AccountFlag的十进制数为112，对应的二进制数为0b1110000。

可以通过以下代码检查是否打开YD_AF_NotifyOrderAccept功能：

```
if (myAccount->AccountFlag & YD_AF_NotifyOrderAccept) {
  // server notification enabled
}
```

账户层面可以控制的功能清单如下：

开关值	说明
YD_AF_SelectConnection	是否可以把席位优选结果上传到柜台供所有人使用，详情参见席位优选。

开关值	说明
YD_AF_AutoMakeCombPosition	**不建议使用。**是否由经纪商的ydAutoCP帮助组合，详情参见自动工具。
YD_AF_BareProtocol	是否允许使用裸协议报单，详情参见裸协议。
YD_AF_DisableSelfTradeCheck	是否关闭自成交检查，详情参见自成交检查。
YD_AF_NotifyOrderAccept	是否需要柜台发送柜台回报，详情参见报单回报。
YD_AF_NoCloseFrozenOnInsertOrder	平仓时是否立即冻结持仓，详情参见紧急平仓。
YD_AF_OrderRefCheck	是否检查OrderRef的单调递增性，详情参见报单号单调递增检查。

3.2.3.7. 日初持仓

日初持仓包括衍生品持仓和现货持仓两种，衍生品持仓即为期货、期权的持仓，现货持仓为沪深300ETF等现货品种的持仓，由于日初持仓的基本用法就是遍历，以便基于日初持仓计算实时持仓，因此易达只提供了遍历所有日初持仓的方法。

```
// 衍生品
virtual int getPrePositionCount(void)=0
virtual const YDPrePosition *getPrePosition(int pos)=0

// 现货
virtual int getSpotPrePositionCount(void)=0
virtual const YDSpotPrePosition *getSpotPrePosition(int pos)=0
```

遍历所有持仓的方法如下所示，下面的方法以衍生品持仓为例，现货的方法是相同的：

```
for(int i = 0; i < pApi->getPrePositionCount(); i++) {
  YDPrePosition *prePosition = pApi->getPrePosition(i);
}
```

衍生品日初持仓和现货日初持仓完全是静态数据。

衍生品日初持仓的结构如下所示：

字段	说明
m_pAccount	持仓所属账户
m_pInstrument	持仓合约
PositionDirection	持仓方向
HedgeFlag	投保标志

字段	说明
PrePosition	日初持仓量
PreSettlementPrice	昨结算价
AverageOpenPrice	日初开仓均价

现货日初持仓的结构如下所示：

字段	说明
m_pAccount	持仓所属账户
m_pInstrument	持仓合约
Position	日初持仓量
ExchangeFrozenVolume	交易所冻结数量
ExecAllocatedVolume	净行权分配量，正值代表收券，负值代表交券
ExecAllocatedAmount	净行权冻结金额，负值代表冻结，需要从可用中扣除，不可能为正值

> 按照《中国证券登记结算有限责任公司关于上海证券交易所股票期权试点结算规则》第55条第二款规定：
>
> 本公司根据检查结果，按照“按比例分配”和“按尾数大小分配”原则，对有效行权与被行权方进行行权指派，将行权指派结果发送结算参与人，并在认沽期权行权方合约账户对应的证券账户中锁定行权交割所需合约标的。**被行权方结算参与人被指派合约对应的维持保证金不予释放。**
>
> 按照《中国证券登记结算有限责任公司关于上海证券交易所股票期权试点结算规则》第58条第一款规定：
>
> 应付合约标的结算参与人无法交付合约标的的，**本公司对未交付部分按照合约标的当日收盘价的110%进行现金结算。**本公司另有规定的除外。

根据监管要求，E+1日易达计算净行权分配金额的公式为，E为行权日：

$$净行权分配金额 = min(净行权交收金额 + 净差券违约金, 净维持保证金)$$

上述公式中各项定义如下：

- 净行权交收金额是依据E日盘后配对结算结果计算的原始交收金额，正值代表收钱，负值代表交钱
- 净差券违约金是为预防交券违约而预先冻结的资金，负值代表需要冻结违约金，0为无违约金，不可能为正值。在E日盘后的配对结算结果中，
  $差券数量 = max((应付券 - 应收券) - (总券数 - 冻结券数), 0)$，则
  $净差券违约金 = -差券数量 \times 现券E日收盘价 \times (1 + 10\%) \times (1 + 10\%)$，其中第一个10%是监管要求，第二个10%在违约情况下无法确定是哪个合约违约，因此只能按照上限进行冻结

- 净维持保证金为期权合约在E+1日盘中按照监管要求冻结的资金，负值代表需要冻结保证金，0为无保证金，不可能为正值

3.2.3.8. 日初组合持仓

日初组合持仓是静态数据，理论上它也应该在静态数据传输阶段，也就是notifyFinishInit之前传输，但是为了兼容老版本API，实际上它是紧跟在notifyFinishInit之后、notifyCaughtUp之前传输到客户端的，虽然传输时间点与其他静态数据不同，但是与其他静态数据一样，日初组合持仓只会在首次登录时推送一次，也就是有notifyFinishInit回调时会被推送到客户端，不会因为断线等原因导致重复推送。为了概念的正确性和便于理解，日初组合持仓的介绍与其他静态数据放在一起。

API没有提供查询接口，因此只能从notifyCombPosition回调函数中收集所有日初组合持仓。

```
virtual void notifyCombPosition(const YDCombPosition *pCombPosition,const YDCombPositionDef *pCombPositionDef,const YDAccount *pAccount)
```

YDCombPosition结构体里面的字段都是静态数据，使用时需要结合该回调中其他参数一起使用：

参数	字段	说明
pAccount		持仓所属账户
pCombPositionDef		组合持仓定义
pCombPosition	Position	持仓量
	CombPositionDetailID	组合持仓明细ID，只对上交所、深交所有意义

3.2.3.9. 系统参数

系统参数是在计算业务过程中会使用到的参数，目前只用于保证金计算。

易达提供了遍历和查找的两组方法。

```
virtual int getSystemParamCount(void)=0
virtual const YDSystemParam *getSystemParam(int pos)=0
virtual const YDSystemParam *getSystemParamByName(const char *name,const char *target)=0
```

前两个方法可以遍历所有系统参数，具体方法如下面代码所示：

```
for(int i = 0; i < pApi->getSystemParamCount(); i++) {
  YDSystemParam *param = pApi->getSystemParam(i);
}
```

第三个方法可以指定查找系统参数，查找调用示例如下所示：

```
getSystemParamByName("MarginBasePrice", "Futures")
```

YDSystemParam都是静态数据，其结构如下：

字段	说明
Name	参数名称
Target	参数的适用对象
Value	参数值

上述Name和Target都是字符串，可能的搭配组合如下表所示：

Name	Target	Value
MarginLowerBoundaryCoef	MarginCalcMethod1	股指期权保证金算法中的保障系数，浮点数，缺省值0.5
MarginBasePrice	Futures或者Options	计算期货或者期权空头的保证金时使用的基础价格，可以选择的值如下： 0：昨结算价 1：开仓价 2：最新价 3：市场平均成交价 4：最新价和昨结算价之间的较大值 目前的常用规则是期货为1，期权为4
OrderMarginBasePrice	Futures或者Options	计算期货或者卖出期权的开仓报单的冻结保证金时使用的基础价格，可以选择的值如下： 0：昨结算价 5：报单价（对于市价单，使用涨停板） 7: 使用MarginBasePrice的价格 目前的常用规则是期货为5，期权为0 注意，期权执行申请报单中冻结的保证金永远是基于昨结算价的

Name	Target	Value
MarginBasePriceAsUnderlying	Options	计算卖出期权保证金时使用的基础产品的价格，可以选择的值如下： 0：昨结算价 2：最新价 4：最新价和昨结算价之间的较大值 目前的常用规则是0
SellOrderPremiumBasePrice	Options	计算卖出开仓期权时反向冻结权利金使用的基础价格，可以选择的值如下： 0：昨结算价 5：报单价（对于市价单，使用跌停板） 6：不反向冻结 目前的常用规则是6 注意，买入开仓期权时冻结的权利金永远是报单价（对于市价单，使用涨停板）
PortfolioMarginConcession	DCELongOptionPortfolio	对于大商所和广期所包含期权多仓的组合持仓，是否优惠计算保证金。可以选择的值如下： 0：不优惠 1：优惠。由于易达平仓时不会检查资金，因此极端情况下可能会导致买期权组合平仓时多收保证金而造成资金透支。
UseCollateral	Account	将质押品获得的质押资金计入昨权益供开仓使用。可以选择的值如下： 0：关闭 1：启用
MarketMaker	System	柜台是否支持做市商报价业务 yes：支持 no：不支持
Test	System	柜台是否是测试柜台，测试柜台不包含性能优化，不可以用于生产
ServerVersion	System	柜台版本号，版本号规范详见版本规则

Name	Target	Value
ConnectionMode	System	柜台的席位模式： 0：全管理席位 1：首席位管理其他非管理 2：全非管理席位 3：全管理作为非管理席位使用，只能用于上期所和中金所
MinApiVersion	System	柜台支持的最低API版本号，版本号规范详见版本规则
MaxApiVersion	System	柜台支持的最高API版本号，版本号规范详见版本规则
MinSelectConnectionGap	System	上报席位优选结果的最小时间间隔，单位为毫秒
MaxSelectConnectionGap	System	上报席位优选结果的最大有效时间，单位为毫秒
MissingOrderGap	System	被系统判定为超时未知单的超时时间，单位为毫秒。
UDPTradingPort	System	UDP报单端口。若柜台没有开启UDP服务，则为0。
XTCPTradingPort	System	XTCP报单端口。若柜台没有开启XTCP服务，则为0。
TradingSegmentDetail	System	是否开启了详细的交易阶段通告 yes：开启 no：关闭

3.2.3.10. 保证金率和费率设置值

保证金率和费率的设置值是柜台对保证金率的实际设置值，例如要设置全体投资者的产品层面的保证金率，只需要在产品层面设置一条记录即可，易达系统会将这条设置值应用到所有投资者的所有属于该产品的合约上，相比逐条设置更容易供人查看。这些设置值主要供易达界面显示设置值所用，对普通投资者没有实际的意义。如需获取保证金率和费率展开后的结果，请参考账户层级信息。

下面是获取保证金率和费率设置值的方法，仅供参考，不做详细解释。

```
// 保证金率设置值
virtual int getMarginRateCount(void)=0
virtual const YDMarginRate *getMarginRate(int pos)=0

// 手续费率设置值
virtual int getCommissionRateCount(void)=0
virtual const YDCommissionRate *getCommissionRate(int pos)=0
```

3.2.3.11. 账户层级信息

账户层级信息提供了交易所级、产品级、合约级的保证金率、费率、权限和传统风控参数，是专门供投资者获取这些参数的结构体，对于这几类信息的说明如下：

- 费率是静态信息，保证金是动态数据，盘中可调，因此在notifyFinishInit中获得的是日初保证金，费率和保证金率是合约层信息。为方便使用，API提供了直接获取合约层面保证金率、费率的方法，详情请参见保证金和手续费相关内容
- 权限是动态数据，盘中可调，保证金参数在各个层面都有设置，但最终要落实到合约层面。API没直接提供查询合约层面权限的方法，可以按照交易权限中的方法自行实现
- 传统风控参数是静态参数，易达的风控参数可以设置在产品和合约两个层面，例如设置在产品层面的撤单数表示限制该产品中所有合约撤单数的总和，因此，需要分别从产品和合约层面获取风控参数。详情请参见风控参数

账户各层级信息可以直接通过下列函数获取：

```
/// when trader call following 3 functions, pAccount should be NULL
virtual const YDAccountExchangeInfo *getAccountExchangeInfo(const YDExchange *pExchange,const YDAccount *pAccount=NULL)
virtual const YDAccountProductInfo *getAccountProductInfo(const YDProduct *pProduct,const YDAccount *pAccount=NULL)
virtual const YDAccountInstrumentInfo *getAccountInstrumentInfo(const YDInstrument *pInstrument,const YDAccount *pAccount=NULL)
```

YDAccountExchangeInfo的主要字段如下所示，其主键为账户和交易所：

字段	说明
m_pAccount	账户结构体指针
m_pExchange	交易所结构体指针
TradingRight	交易权限，动态数据
IsDedicatedConnectionID	专属席位数组位图，详情参见指定席位

YDAccountProductInfo的主要字段如下所示，其主键为账户和产品：

字段	说明

字段	说明
m_pAccount	账户结构体指针
m_pExchange	交易所结构体指针
TradingRight	交易权限，动态数据
TradingConstraints[YD_MaxHedgeFlag]	投机、套保、套利传统风控参数数组，表示属于该产品的所有合约的风控指标相加后予以风控，详情参见风控参数

YDAccountInstrumentInfo的主要字段如下所示，其主键为账户和合约：

字段	说明
m_pAccount	账户结构体指针
m_pExchange	交易所结构体指针
TradingRight	交易权限，动态数据
TradingConstraints[YD_MaxHedgeFlag]	投机、套保、套利传统风控参数数组，表示只限制该合约风控指标上的阈值，详情参见风控参数
m_pMarginRate[YD_MaxHedgeFlag]	保证金率数组，详情参见保证金
m_pCommissionRate[YD_MaxHedgeFlag]	手续费率数组，详情参见手续费

3.2.3.12. 风控参数

易达系统可以按照技术实现来划分为两类风控参数，分别是传统风控参数以及通用风控参数。

- 传统风控参数是易达早期支持的风控规则，主要是常见的开仓量、成交量、持仓量和撤单笔数的控制，传统风控参数不会继续增加，新的风控规则参数全部在通用风控参数中设置
- 通用风控参数是为了适应越来越灵活复杂的风控规则而引入的风控参数设置方法，其设置结构采用了通用的设置方法，要根据每个风控规则去解读其参数，与在账户层级信息中的最终值不同，通用风控参数并没有将不同维度设置的风控参数做归并，需要投资者根据每个风控规则支持的层次结构自行归并

传统风控参数结构体的字段如下表所示，与YDAccountProductInfo和YDAccountInstrumentInfo中的TradingConstraints[YD_MaxHedgeFlag]相结合，就能获得完整的不同投保标志下的风控参数，如需了解详细的风控规则计算方式请参见风控中的具体风控规则。

字段	说明
OpenLimit	开仓量限制
DirectionOpenLimit[2]	买卖开仓量限制

字段	说明
PositionLimit	持仓量限制
DirectionPositionLimit[2]	多空持仓量限制
TradeVolumeLimit	成交量限制
CancelLimit	撤单笔数限制

通用风控规则的参数可以通过以下遍历方法获取：

```
virtual int getGeneralRiskParamCount(void)=0;
virtual const YDGeneralRiskParam *getGeneralRiskParam(int pos)=0;
```

遍历全部通用风控规则参数的样例代码如下所示：

```
for(int i = 0; i < pApi->getGeneralRiskParamCount(); i++) {
  YDGeneralRiskParam *param = pApi->getGeneralRiskParam(i);
}
```

通用风控参数YDGeneralRiskParam是静态数据，其结构体如下所示，主键为(GeneralRiskParamType，AccountRef，ExtendedRef)：

字段	说明
GeneralRiskParamType	风控规则类型
AccountRef	账户序号，-1表示对所有用户生效，其他值表示对本用户生效
ExtendedRef	扩展序号，根据规则定义可以表示交易所、产品、合约的序号
IntValue1	整型参数1
IntValue2	整型参数2
FloatValue	浮点参数

并非所有的风控规则都使用了上面所有的IntValue1、IntValue2和FloatValue，大部分风控规则只使用了上述规则中的部分参数，请参考具体的风控规则，在风控规则中没有列出的参数值则表示没有使用。

3.2.4. 收取动态数据

易达柜台在推送完成静态数据以后，柜台端已经做好了接受报单的准备，如果此时开始报单柜台就会正常推送回报等动态数据。但若此时柜台尚有客户端离线期间未发送到客户端的动态数据，那么客户此时交易的基础可能是错误的，为了通知投资者已经完成了所有历史动态数据的接收并可以开始正常交易，我们增加了这一阶段并通过回调函数notifyCaughtUp通知投资者。

```
virtual void notifyCaughtUp(void)
```

基本原理是登陆的时候柜台会告诉该投资者最大的流水号，API在接收时会比较当前动态数据的流水号与登录时的最大流水号，若当前流水号超过了登录时的最大流水号，那么就说明流水的追及已经完成。该方法被回调仅说明追上了登录时的流水，实际还可能存在登录后产生的流水。如果发生了断线重连，会再次获得该消息，详情请参见交易重连。

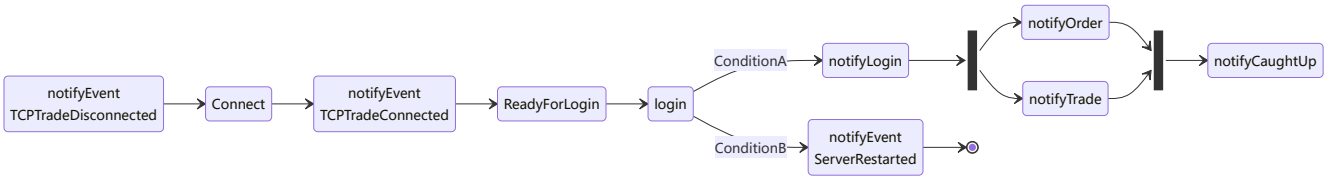
本阶段内收取的动态数据与正常交易阶段收取的数据是一样的，因此在此就不再列举收取动态数据的回调函数，请参考对应业务中对于回调函数的说明。

3.3. 重连

3.3.1. 交易重连

当API探测到与柜台的TCP交易连接中断或者超时后，API会回调notifyEvent(YD_AE_TCPTradeDisconnected)通知交易线程断开了与柜台的连接，此后，API会持续发起到柜台的连接，直到重新连接上后API会回调notifyEvent(YD_AE_TCPTradeConnected)通知，后续的过程与启动过程类似，但是有以下区别：

- 重连过程中不会重新传输静态数据，因为就不会有notifyFinishInit回调，在登陆完成后就开始传输断线过程中发生的报单和成交等流水
- login时不允许换成另外的用户登录，调用login登陆的过程中，API会根据诸多因素判断是否应该继续后续步骤还是终止API，具体逻辑如下：
    - 检查柜台数据指纹是否发生了变化，若发生了变化，则直接进入终止逻辑（ConditionB）。柜台数据指纹是根据日初数据和易达配置信息综合计算出来的，上传了错误的日初数据后重新上传并重启柜台、新的交易日数据被上传并重启柜台、柜台配置的交易所或者席位等技术参数发生了改变并重启柜台，都是常见的导致柜台数据指纹发生变化的原因
    - 如果柜台数据指纹没有变化，则需要继续检查柜台实例ID。每次启动的柜台实例ID都不相同，因此柜台实例ID可以用于判断柜台是否发生了重启。根据柜台是否重启以及API是否启用了高可用功能，存在以下三种可能性：
        - 若柜台没有发生重启，说明此次断线是因为网络问题导致的，系统将正常继续执行后续步骤（ConditionA）
        - 若柜台发生了重启，并且API启用了高可用功能，则会继续正常执行后续步骤（ConditionA），此时在notifyLogin之前会收到notifyEvent(YD_AE_ServerSwitch)回调，表示发生了一次主备切换
        - 若API没有启用高可用功能，则会直接进入终止逻辑（ConditionB）
    - 进入终止逻辑后，通常情况下API会调用exit()系统调用导致整个进程退出，这会连带策略程序一起退出。如果希望避免这种情况，可以在notifyEvent(YD_AE_ServerRestarted)中主动销毁API，以跳过API调用退出指令



综上所述，当柜台在同一个交易日内重启时，易达API为投资者提供了三种不同程度的处置方式：

- 重启进程：这种方式最为干净、简单且不易发生错误，策略程序重新从头接收柜台流水并恢复到最新状态。缺点是策略程序会被连带退出，在生产上可能欠缺可操作性。
- 重建API实例：这种方式相对干净且不会导致策略程序退出，重建实例后API会重新推送所有流水。若策略程序保存且需要复用本地流水，可能需要合并API重新下发的流水，而且策略程序需要确保在重建过程中不可以访问被销毁实例的数据，否则会导致程序奔溃，因此需要对策略程序有较为细腻的控制。同时，重建实例过程中也会遇到策略端超时未知单以及外部系统报单的问题需要处理，详情请参考高可用章节内容。
- 重新建立连接：即开启高可用模式下的方式，对策略程序的影响几乎无感知，发生自动切换后仅仅通过notifyEvent(YD_AE_ServerSwitch)告知策略程序发生了高可用切换。考虑到通常策略程序一般都具有处理策略端超时未知单以及外部系统报单的能力，因此本方式对于策略程序最为友好。本模式下具体情况请参考高可用。

3.3.1.1. 高可用

为了满足做市商和重视可用性的投资者的需要，易达推出了高可用集群功能，它可以确保最大程度上的业务连续性，当发生硬件宕机、柜台程序错误等故障时，投资者的交易会在最短时间内恢复。

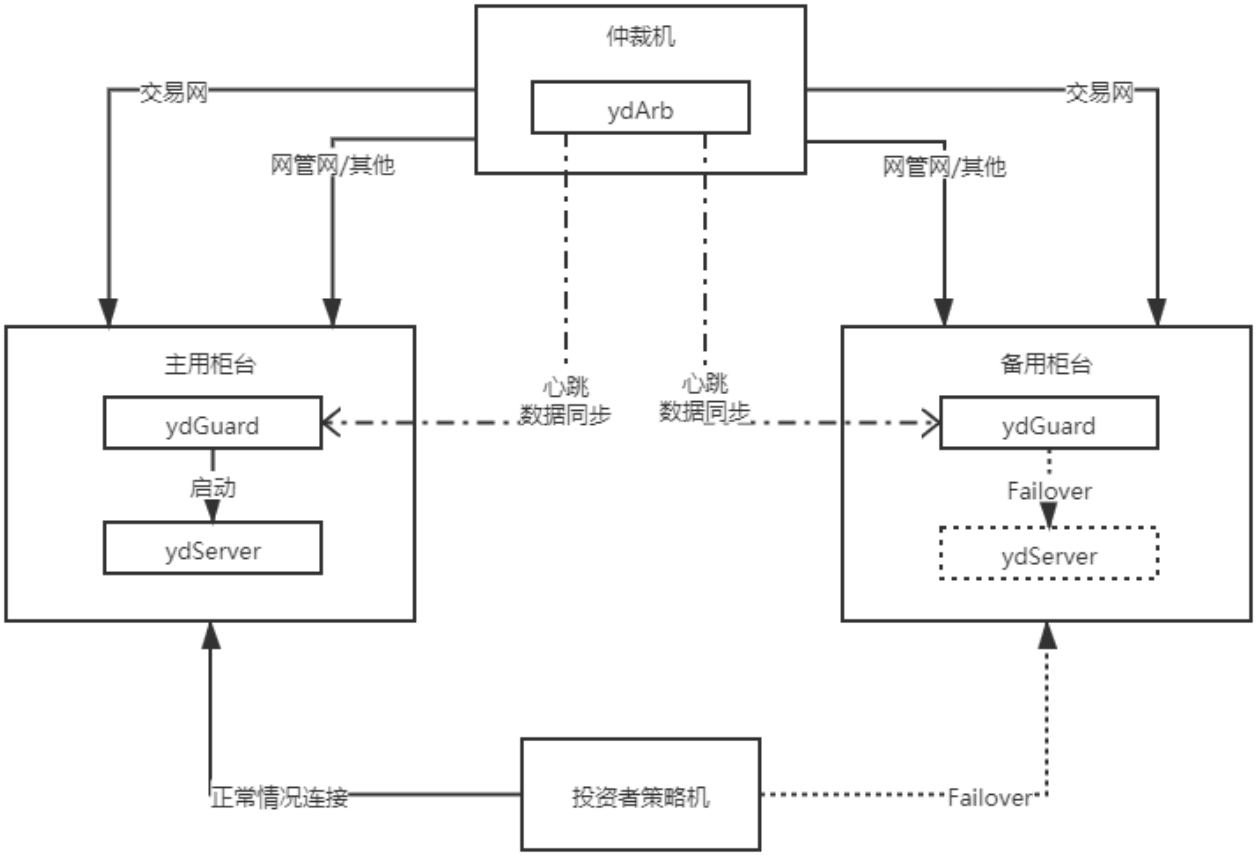
高可用集群由三台服务器组成的热备集群，包括两台运行ydServer的柜台服务器和一台运行ydArb的仲裁服务器。为满足不同用户对于性能和可用性的权衡，主备柜台服务器的硬件配置有多种不同的组合：

- 最高保护：主备服务器均采用普通服务器，最大限度上确保主用服务器极少发生故障，即便发生故障，备用服务器也可以在极大程度上成功启动并接替交易。在当前版本中，经过性能调优的普通服务器的穿透性能比超频机差500ns，对于做市商、自营用户等极端关注系统稳定性和交易可用性的用户较为适合。
- 兼顾性能和保护：主用服务器采用超频机，备用服务器采用普通服务器。考虑到目前超频机较为稳定，通常不会发生故障，因此绝大部分时间交易都在超频机上，可以获得最高的性能，一旦超频机发生故障，由于备机使用的普通服务器极为稳定，备用服务器也可以在极大程度上成功启动并接替交易。这种方案兼顾了性能和可用性，当发生切换时放弃一定的性能但是确保了交易的持续性。
- 最高性能：主备服务器均采用超频机，最大程度上保证了交易性能。但是由于备用服务器和主用服务器的上架时间相同，老化时间也相同，因此当发生主备切换时存在一定的可能性备机无法启动，但是目前超频机的稳定性得到了极大的增强，发生问题的概率不大。这种配置组合适合大部分投资者。

高可用集群的三台服务器均应建立仲裁网，用于传输心跳、交易流水和启停控制指令，仲裁网只需保证仲裁机能连通主备服务器即可，主备服务器之间的连通性没有要求。按照每个席位每天约600MB计算，通常的千兆网络就能满足仲裁网的带宽需求，若网管网容量能满足上述带宽要求，可以使用网管网作为仲裁网。为了避免仲裁网网络故障带来的无效切换，建议将投资者的交易网配置为仲裁网的备份。当仲裁网正常工作时，所有流量将在仲裁网上，交易网不会有任何的流量；当仲裁网故障时，才会将流量切换到交易网。若仲裁机发现无法通过仲裁网连通主用柜台，它将通过备用网络尝试联系主用柜台，如果此时仍然无法连接，此时发起主备切换就显得更为合理，因为这个时候投资者也大概率无法连通柜台。

高可用集群的主要组件功能如下：

- ydArb：部署在仲裁机上面的常驻仲裁程序。通过心跳信息监控集群的可用性，当主用节点发生故障时，主动通知备用节点启动接替主用节点。同时还负责将主用节点上的盘中数据（交易所流水和柜台本地流水）持续传输到备用节点，以保证备用节点最快速启动。
- ydGuard：部署在主备用服务器上的ydServer常驻守护进程。检测ydServer的状态并向ydArb通报，同时主用柜台上ydGuard向ydArb传输盘中数据，而备用柜台上ydGuard从ydArb接收盘中数据。主用柜台ydGuard的数据传输相对ydServer的交易而言是异步的，极有可能发生回报已经发送到投资者但是没有同步到备用柜台的情况。



当要连接到高可用集群时，需要按照如下样例修改API的配置文件。

```
RecoverySiteCount=2
TradingServerIP=<ip of master host>
TradingServerPort=<port of master host>
TradingServerIP2=<ip of slave host>
TradingServerPort2=<port of slave host>
```

正常运行时，主用服务器的ydServer进程处于启动状态，备用服务器的ydServer进程处于停止状态，ydArb和ydGuard构成的集群持续将交易流水经由仲裁机实时从主用柜台传送到备用柜台，此时投资者的API连接到主用服务器上面的ydServer交易。

当发生主备切换时，备用服务器上的ydServer会被拉起，并加载同步的盘中数据，尽可能恢复到主用柜台发生故障时的状态。易达API检测到断线后，将轮询主备服务器的IP地址直到重新连上，连上后的后续步骤可以参考交易重连章节内容。由于主备切换固有的复杂性，以下问题是无法避免的，需要投资者自行处理：

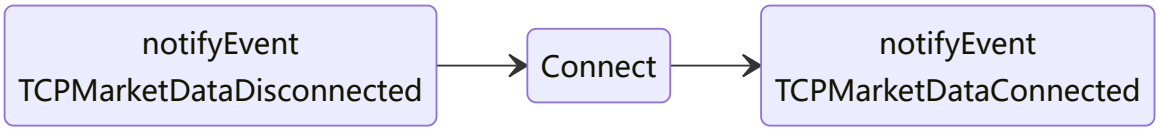
- 本地超时未知单：若柜台发生故障的时间点为报单从策略程序发出之后到在柜台发往交易所之前这个时间段内，策略程序会一直等待回报更新报单状态，但是由于报单没有报送到交易所，因此在柜台和交易所的角度来看，这笔报单并不存在，因此也不可能撤单。因此，策略程序需要建立超时机制，如果遇到主备发生切换的时候，应该主动删除策略端超时的本地报单。请注意，本地超时报单与超时未知单发生的原因、产生的后果是不同的，因此处理方式也是不同的。
- 无法查到对应关系的报单和成交：若柜台发生故障的时间点为报单从柜台发出之后到主用柜台的流水同步到备用柜台之前的这个时间段内，显而易见的是发生故障后，策略程序也没有收到对应的交易所报单回报（否则就不存在需要特殊处理的丢单）。在备用柜台完成启动后，易达会保证不会丢失报单本身，但是由于主用柜台的本地流水还没有同步，备用柜台重新发送的这一报单回报是含有OrderSysID、OrderLocalID以及RealConnectionID的，但不会含有OrderRef的。那么投资者在收到这笔回报后，是无法与策略端记录的报单相关联的，当遇到OrderRef为-1这种情况时，请按照普通的外部报单逻辑一样处理即可，而那笔没有被匹配到的策略端报单则应按照本地超时未知单的处理逻辑进行处理。

单柜台高可用是双柜台高可用版本的特例版本，等同于在双柜台高可用模式下将两个IP地址配置为相同的地址，两者实际效果基本是相同。单柜台高可用模式简化了策略程序与到柜台在同一个交易日内重启时的处理过程，因此我们推荐投资者尽可能使用单柜台高可用特性。单柜台高可用的对应配置文件的配置样例如下：

```
RecoverySiteCount=1
TradingServerIP=<ip of master host>
TradingServerPort=<port of master host>
```

3.3.2. 行情重连

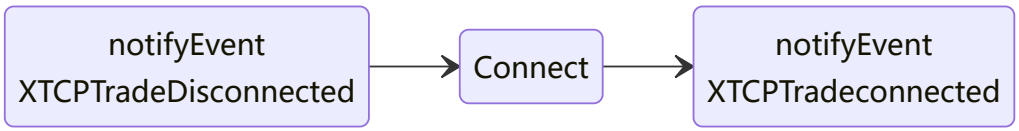
当API探测到与柜台的TCP行情连接中断或者超时后，API会回调notifyEvent(YD_AE_TCPMarketDataDisconnected)通知行情线程断开了与柜台的连接，此后，API会持续发起到柜台的连接，直到重新连接上后API会回调notifyEvent(YD_AE_TCPMarketDataConnected)通知。相比交易重连的过程，行情重连相对简单很多。重连成功后，不需要再次订阅行情，API将会自动重新订阅断线前已订阅合约，并继续收取TCP行情。



3.3.3. XTCP重连

当API探测到与柜台的XTCP交易连接中断或者超时后，API会回调notifyEvent(YD_AE_XTCPTradeDisconnected)通知XTCP线程断开了与柜台的连接。此后，API会持续发起到柜台的连接，直到重新连接上后API会回调notifyEvent(YD_AE_XTCPTradeConnected)通知。

若断线重连期间投资者报单，则API会降级使用普通TCP报单。



3.4. 销毁

策略程序可以主动销毁API实例以避免内存泄漏或者避免API主动退出进程，由于易达API的结构较为复杂，无法通过删除实例的方式清理干净，因此易达在ydApi中提供了startDestroy用于实例清理工作，请永远不要尝试直接删除API实例。

```
1 virtual void startDestroy(void)
```

startDestroy将发起两阶段的异步销毁流程，该函数返回后并不表示清理已经完成。

- 第一阶段的主要工作是尝试关闭所有属于API的连接和线程，完成后将通过YDListener.notifyBeforeApiDestroy通知第一阶段已结束。在此过程中，包括notifyBeforeApiDestroy的回调函数中，所有API中的交易功能都将无法使用，YDListener中的交易回调函数也不会被回调，但是ydApi上所有的查询函数以及ydExtendedApi里管理的数据仍然可以使用。

  ```
  1 virtual void notifyBeforeApiDestroy(void)
  ```

- 第二阶段的主要工作是销毁所有数据，完成后将通过YDListener.notifyAfterApiDestroy通知清理已全部结束。在此过程中，包括notifyAfterApiDestroy的回调函数中，所有API的功能以及从API中获得的指针均无法使用。

  ```
  1 virtual void notifyAfterApiDestroy(void)
  ```

清理结束后，请自行清理您实现并实例化的YDListener子类实例，本函数不会帮您销毁该实例。在策略程序中，您可以重新创建新的API实例。

4. 资金

4.1. 权益

4.1.1. 昨余额

昨余额又被称为日初权益，主要由昨权益组成，期货交易所和证券交易所中昨余额的计算方法并不相同，以下将分别介绍。

> 为了便于经纪商配置使用易达系统，减少不必要的配置错误，易达直接使用现有主席系统的日初文件经过转换后作为易达的日初数据，包括昨余额、日初持仓、保证金率、费率、部分风控规则等。目前易达的期货交易所日初数据可以从CTP的syncmerge或者CTP异构数据接口中直接读取；而证券交易所的股票期权日初数据是从多个数据源获得的，为了满足需求并确保多数据源的勾稽关系，易达引入了一些内部计算。如果经纪商使用的主席柜台是其他品牌的，易达将不断增加新的日初数据转换程序，以满足投资者不受限制的在经纪商分仓参与交易的需求。

昨余额可以在YDAccount.PreBalance中获取。

4.1.1.1. 期货交易所昨余额

期货交易所的数据是从CTP的syncmerge或者CTP异构数据接口中读取后简单计算获得的，其计算公式如下。其中昨权益、质押金额、货币质入和质出金额是直接从源数据中直接读取的，固定扣减是在易达系统中设置的。

$$昨余额 = 昨权益 + 质押金额 + 货币质入金额 - 货币质出金额 - 固定扣减_{易达}$$

目前市场上主流的做法是经纪商与投资者签订协议，约定了在每个次席柜台上的固定权益，由经纪商修改日初数据从而达到资金切割的作用，这种做法虽然简单，但是当发生某个交易所市场行情来临时，可能会导致因为资金缺少而没能抓住的行情，即便是可以通过出入金的方式来调度资金，但是人工操作的复杂审核会耽误不少交易机会。因此易达在设计之初就考虑了这种情况，当投资者在多套易达柜台同时交易时，我们希望经纪商不要对投资者的昨权益进行拆分，每个柜台上读取的昨权益都是相同的，然后通过资金使用限度这个参数来控制投资者在各个柜台的昨权益占比，结合易达资金调拨系统，在满足同一投资者在所有柜台的资金使用限度加总小于等于100%的前提下，投资者在盘中可以自行修改资金使用限度以实现实时资金调拨的效果，关于资金使用限度的含义请参见今余额。

固定扣减是易达为方便同时在易达和其他次席柜台交易的投资者也能在易达柜台中使用资金使用限度功能，如下表所示，假设某投资者共有100万权益，在易达柜台中分配80万权益，在其他次席柜台中分配了20万权益，那么在使用了固定扣减的情况下，资金使用限度功能仍然能正常工作。

交易所	柜台类型	柜台类型总权益	资金使用限度	占用权益
SHFE	易达	80	40%	32
DCE	易达	80	50%	40
CFFEX	易达	80	10%	8

交易所	柜台类型	柜台类型总权益	资金使用限度	占用权益
CZCE	其他	20		20

4.1.1.2. 证券交易所昨余额

证券交易所的昨权益可以与期货交易所一样直接读取投资者在沪深两市的昨权益，然后通过资金使用比例进行动态拆分，这是易达推荐的做法。同时为了满足当前市场上投资者的习惯和实际需要，易达也支持按照每个投资者层面定设置的可用资金拆分比例（下称拆分比例，在后台定义，无法在API上读出）拆分，要实现该功能需要从监控中心cfmmc上报文件（只包含沪深两市数据）、中登上海、中登深圳三个数据源中通过计算后获取的，整个计算过程较为复杂，虽然易达无法保证在按照可用资金拆分后易达柜台中显示的可用资金与主席柜台上按照拆分比例计算后的结果相同，但易达一定保证金沪深两市的昨权益相加始终等于主席柜台的昨权益，想了解易达具体计算方法的投资者可以继续阅读，不感兴趣的投资者可以直接跳过以下内容。

在行权交割日由于监控中心cfmmc上报文件（只包含沪深两市数据）、中登上海、中登深圳三份数据的勾稽关系存在差异，因此需要首先计算出适用于易达系统的可用资金作为拆分基础。

$$\text{可用资金}_{\text{易达}} = \text{昨权益}_{cfmmc} - \text{沪市保证金}_{cfmmc} - \text{深市保证金}_{cfmmc} + min(\text{净行权分配资金}_{\text{中登上海}}, 0) + min(\text{净行权分配资金}_{\text{中登深圳}}, 0)$$

然后根据计算得到的可用资金进行拆分就可以得到易达系统中的昨权益。

$$\text{沪市昨权益}_{\text{易达}} = \text{沪市拆分比例}_{\text{易达}} \times \text{可用资金}_{\text{易达}} + \text{沪市保证金}_{cfmmc} - min(\text{净行权分配资金}_{\text{中登上海}}, 0)$$

$$\text{深市昨权益}_{\text{易达}} = \text{深市拆分比例}_{\text{易达}} \times \text{可用资金}_{\text{易达}} + \text{深市保证金}_{cfmmc} - min(\text{净行权分配资金}_{\text{中登深圳}}, 0)$$

简单联立三个公式后能看到，易达系统中的昨权益加总等于监控中心文件中的昨权益。

$$\text{昨权益}_{cfmmc} = \text{沪市昨权益}_{\text{易达}} + \text{深市昨权益}_{\text{易达}}$$

与期货交易所类似，固定扣减功能仍然是生效的，但是如果其他次席柜台系统是按照可用拆分的，那么固定扣减是无法使用的，因此作用不大，通常情况下固定扣减为0。

$$\text{昨余额} = \text{昨权益} - \text{固定扣减}_{\text{易达}}$$

4.1.2. 今余额

在易达的设计中，静态权益描述了投资者层面的当日整体权益，该投资者所在任何易达柜台上的静态权益都是相同的，其中入金和出金的金额也是在投资者层面的，即只要在该投资者在主席柜台上出入金，那么该投资者所在任何易达柜台上都会对应入金或者出金，且金额与主席柜台相同。这样就为引入资金使用限度的概念做好了准备。

如果经纪商自行按照固定金额拆分或者按照可用资金比例拆分，那么入金和出金就表达为针对单套柜台的出入金金额，静态权益也仅仅表示该投资者在对应柜台中的当日权益，而且在这种情况下不应该使用资金使用限度，即资金使用限度保持为100%，否则会造成资金错乱。

使用ydExtendedApi的投资者可以调用YDExtendedAccount.staticCashBalance()获取静态权益。

$$\text{静态权益} = \text{昨余额} + \text{入金} - \text{出金}$$

无论在何种使用模式中，今余额都表示当前投资者在对应柜台中的当日权益。如果按照易达原始设计来使用柜台，那么就可以使用资金使用限度来完成自动出入金同步以及投资者自主的资金调拨功能。

使用ydExtendedApi的投资者可以在YDExtendedAccount.Balance获取今余额。

$$今余额 = (静态权益 - 冻结出金) \times 资金使用限度$$

4.1.2.1. 资金使用限度

易达支持通过设置资金使用限度分隔多套系统的日初权益，只要每套柜台的资金使用限度加总不超过100%，就可以保证金不会透支。资金使用限度的设置值可以从YDAccount.MaxMoneyUsage中获取。

在经纪商没有拆分资金且合理设置多套易达柜台的资金使用限度的条件下，投资者可以实现自动出入金和自主资金调拨的功能，下面我们通过一个例子来说明。

例如，某投资者同时在中金所、大商所、上期所三套易达系统中交易，其静态权益为100万元，资金使用限度分别为50%、30%和20%。为求简单，例子中忽略了保证金的影响，在任何情况下若出金和设置资金使用限度会导致可用资金不足的都会被系统拦截。因此，默认以下操作都不触发该限制。

	中金所	大商所	上期所
初始资金使用限度	50%	30%	20%
今余额	50	30	20
入金100万后的今余额	100	60	40
上期所需紧急调拨资金，设置后的资金使用限度	10%	20%	70%
调拨后今余额	20	40	140
出金50万后的今余额	15	30	105

资金同步系统ydSync可以即时读取CTP风控系统的出入金流水然后同步到所有易达柜台；资金调拨系统ydFundManager实现了多套易达柜台的资金调拨功能，经纪商可以给投资者开立调拨账户以便投资者登录资金调拨系统自行调拨资金。如有需要请联系经纪商安装，但是上述功能的前提条件是经纪商没有做资金拆分。

4.1.3. 其他权益

为了最终计算资金权益和市值权益，我们把资金权益和市值权益公式中与交易有关的部分称为动态权益。动态权益只与柜台所在交易所有关，两套部署在相同交易所的拥有管理席位的柜台对于同一投资者的动态权益是相同的，但是若上述的两套中的一套配置的是全非管理席位的，这套全非管理席位的柜台无法收到其他柜台的成交数据，此时只有那套拥有管理席位的柜台的动态权益是准确的。当然，如果该交易所只有一套柜台，那么无论该使用的是何种席位，其动态权益总是准确的。单套柜台的动态权益的计算公式如下所示：

$$动态权益 = 期货平仓盈亏 + 期货持仓盈亏 + 净权利金 - 手续费$$

在动态权益概念的帮助下，单套柜台的资金权益和市值权益的计算公式如下所示，基于上述原因，单套柜台的资金权益和市值权益也同样具有简单可加性：

$$今资金权益 = 静态权益 \times 资金使用限度 + 动态权益$$

$$昨资金权益 = 昨余额 \times 资金使用限度$$

$$今市值权益 = 今资金权益 + 今期权市值$$

$$昨市值权益 = 昨资金权益 + 昨期权市值$$

上述权益只能在ydExtendedApi中获取：

- 动态权益：调用YDExtendedAccount.dynamicCashBalance()获得
- 今资金权益：调用YDExtendedAccount.cashBalance()获得
- 昨资金权益：调用YDExtendedAccount.preCashBalance()获得
- 今市值权益：调用YDExtendedAccount.marketValue()获得
- 昨市值权益：调用YDExtendedAccount.preMarketValue()获得

4.2. 可用资金

$$\begin{aligned}可用资金 = 今余额 + 期货平仓盈亏 + min(期货持仓盈亏, 0) + 净权利金 - 手续费 \\ -期货保证金 - 卖出期权保证金 - 期权行权保证金 - 净行权冻结金额\end{aligned}$$

使用ydExtendedApi的投资者可以调用YDExtendedAccount.usable()获取可用资金。按照监管规定：持仓盈利不可开仓，持仓亏损要扣除。因此，易达在计算可用资金时扣除了持仓亏损。

> YDExtendedAccount.Available与可用资金比较相像，从下列公式可以看出Available并没有包括持仓亏损，因此Available是不能等同于可用资金来使用的。
>
> $$\begin{aligned}Available = 今余额 + 期货平仓盈亏 + 净权利金 - 手续费 \\ -期货保证金 - 卖出期权保证金 - 期权行权保证金 - 净行权冻结金额\end{aligned}$$

由于API与柜台刷新的时机以及刷新时使用的价格并不完全相同，可能会导致在API上显示资金足够的但是被柜台因资金不足而拒绝的情况发生。一般这类问题会发生在资金利用率较高的情况下，如果您希望确认因资金不足而被柜台拦截的单子的具体情况，可以联系经纪商从柜台日志中获取拒单时的资金情况，请注意其中Available并不是可用资金，需要扣除PositionProfit的亏损部分才行。

可用资金和Available需要刷新重算后才能显示根据最新价计算的结果，具体刷新方式请参见资金刷新机制。

4.3. 出入金

当投资者有出入金发生时，可以通过YDAccount.Deposit获得累计入金金额，通过YDAccount.Withdraw获得累计入金金额，YDAccount.Deposit和YDAccount.Withdraw在一个交易日内是单调递增的，YDAccount.Deposit-YDAccount.Withdraw即可获得当日出入金净额。

当前冻结出金金额可以通过YDAccount.FrozenWithdraw获得，冻结出金金额会随着经纪商执行冻结出金操作而不断累加，经纪商解除冻结时YDAccount.FrozenWithdraw会相应减少，或者经纪商正式出金时也会相应扣减被与出金金额相同的冻结金额。

当发生出入金或者出金冻结时，会通过notifyAccount通知到投资者，由于YDAccount其他字段发生变化时也会通知到投资者，因此投资者需要在程序中检测上述三个字段以确认是否发生了出入金。

```
virtual void notifyAccount(const YDAccount *pAccount)
```

4.4. 权利金

当日权利金可以从YDExtendedAccount.CashIn读取，期权买方付出权利金时减少当日权利金，期权卖方获得权利金时增加权利金，报单冻结权利金和成交权利金合并计算在该字段之中，不区分冻结权利金和权利金。

期权报单从柜台发出后，将根据报单权利金的计算方式在YDExtendedAccount.CashIn中增减，随着报单被逐步成交直至全部成交，或者报单被撤单，因为报单而产生的冻结权利金将会被逐步扣减至0，同时，权利金也会随着成交的产生而相应增减。

4.4.1. 报单权利金

期权买方报单权利金的计算公式为：

$$期权买方报单权利金 = -价格 \times 合约乘数 \times 报单数量$$

其中，当报单为市价单时，价格为涨停板价，当报单为限价单/FAK/FOK时，价格为报单价格。

期权卖方报单权利金的计算公式为：

$$期权卖方报单权利金 = 价格 \times 合约乘数 \times 报单数量$$

其中，价格受到YDSystemParam中（Name, Target）为（SellOrderPremiumBasePrice, Options）的参数值控制，以下列举不同参数值时系统的处理方式：

- YD_CBT_PreSettlementPrice：昨结算价
- YD_CBT_OrderPrice：当报单为市价单时，价格为跌停板价，当报单为限价单/FAK/FOK时，价格为报单价格
- YD_CBT_None：不增加权利金，等同于价格为0，即期权卖方在报单时不会获得权利金。这是默认配置的方式，也是主流系统当前使用的方式。

4.4.2. 成交权利金

期权买方成交权利金的计算公式为：

$$期权买方成交权利金 = -成交价 \times 合约乘数 \times 报单数量$$

期权卖方成交权利金的计算公式为：

$$期权卖方成交权利金 = 成交价 \times 合约乘数 \times 报单数量$$

4.5. 期权市值

$$昨期权市值 = 期权买持仓量 \times 昨结算价 \times 合约乘数 - 期权卖持仓量 \times 昨结算价 \times 合约乘数$$

$$今期权市值 = 期权买持仓量 \times 最新价 \times 合约乘数 - 期权卖持仓量 \times 最新价 \times 合约乘数$$

昨期权市值和今期权市值分别可以通过YDExtendedAccount.PrePositionMarketValue和YDExtendedAccount.PositionMarketValue获取。

今期权市值需要刷新后才能显示根据最新价计算的结果，具体刷新方式请参见资金刷新机制。

4.6. 手续费

报单手续费在报单成功后收取，撤单手续费在撤单成功后收取，成交手续费在成交的时候收取，易达柜台不会在报单发出后冻结成交手续费。

$$报单手续费 = 报单笔数 \times 报单每笔手续费$$

$$撤单手续费 = 撤单笔数 \times 撤单每笔手续费$$

$$开仓成交手续费 = 开仓量 \times (开仓成交手续费率 \times 成交价格 \times 合约乘数 + 开仓成交每手手续费)$$

$$平今仓成交手续费 = 平今仓量 \times (平今仓成交手续费率 \times 成交价格 \times 合约乘数 + 平今仓成交每手手续费)$$

$$平昨仓成交手续费 = 平昨仓量 \times (平昨仓成交手续费率 \times 成交价格 \times 合约乘数 + 平昨仓成交每手手续费)$$

$$手续费 = \sum_{所有报单} 报单手续费 + \sum_{所有撤单} 撤单手续费 + \sum_{所有开仓成交} 开仓成交手续费 + \sum_{所有平今成交} 平今仓成交手续费 + \sum_{所有平昨成交} 平昨仓成交手续费$$

上述手续费率可以通过ydApi中的getInstrumentCommissionRate，函数签名如下所示。查询时需要指定合约和投保标志可以唯一确定一个保证金率。

```
virtual const YDCommissionRate *getInstrumentCommissionRate(const
YDInstrument *pInstrument,int hedgeFlag,const YDAccount *pAccount=NULL)=0;
```

返回的保证金率的对应关系如下表所示：

参数	字段	说明
YDCommissionRate	OpenRatioByMoney	开仓成交手续费率
	OpenRatioByVolume	开仓成交每手手续费
	CloseRatioByMoney	平昨仓成交手续费率
	CloseRatioByVolume	平昨仓成交每手手续费
	CloseTodayRatioByMoney	平今仓成交手续费率
	CloseTodayRatioByVolume	平今仓成交每手手续费
	OrderCommByVolume	报单每笔手续费
	OrderActionCommByVolume	撤单每笔手续费

不同交易所平仓手续费的计算方法有所区别，我们在YDExtendedPosition.YDPositionForCommission上维护了该持仓用于手续费计算的昨仓数量，投资者可以通过计算YDExtendedPosition.Position-YDExtendedPosition.YDPositionForCommission获得用于手续费计算的平今仓数量。

为方便投资者分辨对应合约优先平今还是平昨，投资者可以通过YDExchange.CloseTodayFirst获得该合约对应交易所是否优先平今。

下面简要说明各个交易所在计算平仓手续费时，使用的今仓和昨仓持仓明细的顺序：

> 请注意用于手续费计算的持仓明细、用于计算平仓盈亏的持仓明细以及用于计算组合保证金的持仓明细都是单独维护的，因此请根据实际使用场景选择正确的持仓信息

- 对于上期所和能源所，根据平今或者平昨指令决定
- 对于中金所、大商所、广期所和郑商所，优先平今仓
- 对于上交所和深交所，优先平昨仓

上交所卖开成交不收取手续费，与手续费参数设置无关。

4.7. 资金刷新机制

易达保证金、持仓盈亏和持仓市值并不是从柜台查询而来，而是直接在客户端使用与柜台相同的方法计算得来，因此，需要在盘中实时刷新保证金和持仓盈亏，以使得可用资金与柜台实际保持一致。

自从`1.98`起易达API提供了多种盘中自动刷新保证金率和持仓盈亏的机制，对于使用不同API的投资者给予了不同程度的支持。通过设置API配置文件中的RecalcMode参数可以指定API的三种刷新机制，分别是关闭、只订阅行情和自动三种模式，下面我们分别介绍这三种机制。

4.7.1. 关闭模式

关闭即完全关闭自动刷新机制，需要由投资者自行决定刷新的时机和方式。

对于使用ydApi的投资者，由于ydApi并没有计算资金和持仓，因此，所有的刷新工作就完全由投资者自行进行，API完全不参与整个过程。

对于使用ydExtendedApi的投资者，当投资者需要刷新时，可以通过调用recalcMarginAndPositionProfit刷新保证金和持仓盈亏，调用recalcPositionMarketValue刷新持仓市值，这两个函数均无输入参数。由于关闭模式不会自动订阅行情，请务必保证接收TCP行情（ConnectTCPMarketData=yes），否则将始终按照日初行情计算刷新。

4.7.2. 订阅行情模式

订阅行情模式可以帮助投资者自动订阅并刷新当日所有交易过的合约或者持仓合约的行情，如果自动订阅了一个期权合约，就会自动订阅其基础合约，以便准确计算期权保证金。但是自动订阅的行情并不会直接通过notifyMarketData发送给投资者，如果需要通过notifyMarketData收取的，请使用subscribe单独订阅。本模式对使用ydApi和ydExtendedApi的投资者均有帮助。

对于使用ydApi的投资者，行情到达时会刷新API中YDMarketData行情，投资者可以直接读取对应合约的价格来自行刷新。

对于使用ydExtendedApi的投资者，行情会保存在ydExtendedApi之中，当投资者需要刷新时，可以通过调用recalcMarginAndPositionProfit刷新保证金和持仓盈亏，调用recalcPositionMarketValue刷新持仓市值，这两个函数均无输入参数。

4.7.3. 自动模式

自动模式包含了订阅行情模式的所有功能，并在此基础上，针对不同类型的API提供了更进一步的支持。

对于使用ydApi的投资者，为了能帮助投资者错开可能的报单时间，API会通过notifyRecalcTime通知投资者系统检测到的安全刷新时间，投资者可以在这个函数中调用自己的刷新方法。

安全刷新时间是依据上一次收到的TCP行情时间，并结合API配置文件中的RecalcMarginPositionProfitGap和RecalcFreeGap的设置值计算出来的，其中RecalcMarginPositionProfitGap是指连续两次刷新的最小间隔，最小可以被设置为1000毫秒，低于1000毫秒的设置将会被调整为1000毫秒，RecalcFreeGap是指安全刷新时间与行情到达前后的安全距离，其设置区间为0至100毫秒，所有超出该范围的设置值都会被调整为距离该值最近的合法值。

具体计算方法为，当距离上次安全刷新时间达到RecalcMarginPositionProfitGap之后，以API上次收到的TCP行情为0毫秒，以250毫秒长度为一个检测段，在该检测段内，去除该检测段前后RecalcFreeGap毫秒后剩余的时间段的开始时间即为安全刷新时间。假设RecalcFreeGap为100毫秒，收到行情的时点是0毫秒，可行的时间段100毫秒至150毫秒共50毫秒的时间段，而安全刷新时间即为100毫秒。

考虑到行情会持续到达，导致安全刷新时间可能会被持续错过，因此我们设置了一个保护时间，即距离上次刷新时间超过RecalcMarginPositionProfitGap的3倍时间还没有刷新的话，ydApi则会通过notifyRecalcTime通知客户强制刷新。

对于使用ydExtendedApi的投资者，API会在安全刷新时间自动调用recalcMarginAndPositionProfit和recalcPositionMarketValue刷新，无需投资者编写任何代码，此时，notifyRecalcTime会在调用完成上述两个刷新函数后通知到投资者，投资者可以自行决定是否需要在该时点做除了刷新之外的其他工作。

易达已经尽可能帮助投资者避开可能的报单时间，但是有可能投资者的报单行为超出了易达的设计预期，因此，如果您发现使用自动模式的刷新时间与报单时间冲突的，可以考虑调整参数或者使用订阅行情模式，自行寻找刷新时间。

5. 持仓

5.1. 持仓结构

各个交易所管理持仓的方式是不同的，有的交易所严格区分了今仓和昨仓并有专门平今和平昨指令；有的交易所并不严格区分今仓和昨仓，也没有专门的平今或者平昨指令，只是在个别业务上（例如平今仓手续费）有区分，因此，易达根据交易所的持仓结构实现了相应的持仓结构。为了便于投资者区分当前交易所使用的持仓结构，投资者可以查看YDExchange.UseTodayPosition的值：当该值为True时则表示该交易所区分了今仓和昨仓，易达系统将会用两条持仓来对应今仓和昨仓；当该值为False时则表示该交易所不区分今仓和昨仓，易达系统只用一条持仓来表示对应的持仓。

YDExtendedApi的持仓组织方式与易达柜台内部的组织方式基本一致，因此下面将以YDExtendedApi的持仓YDExtendedPosition为例，介绍在上述两种持仓结构中用于计算手续费、平仓盈亏和组合业务的规则和相关逻辑。YDExtendedPosition的与持仓结构有关的字段如下所示，主键为合约、持仓日期、持仓方向和投保标志。

字段	说明
getInstrument()	合约
PositionDate	持仓日期 YD_PSD_History：昨仓 YD_PSD_Today：今仓 若交易所不区分今仓和昨仓，一律使用YD_PSD_History
PositionDirection	持仓方向 YD_PD_Long：多头持仓 YD_PD_Short：空头持仓
HedgeFlag	投保标志，期货交易所与股票期权交易所的定义不同。 期货交易所如下： YD_HF_Speculation=1：投机 YD_HF_Arbitrage=2：套利，只有中金所支持。为便于编写程序，易达为是否支持套利交易和持仓提供了参数化表示，YDExchange.UseArbitragePosition为true时表示该交易所支持套利交易和持仓，否则为不支持 YD_HF_Hedge=3：套保 股票期权交易所如下： YD_HF_Normal=1：普通 YD_HF_Covered=3：备兑

字段	说明
Position	当前主键下的总持仓量 若区分今仓和昨仓，则分别表示合约在某持仓方向、某投保标志上的今仓和昨仓的持仓量 若不区分今仓和昨仓，则表示合约在某持仓方向、某投保标志上的总持仓量
PositionDetailList	当前主键下的用于盈亏计算的持仓链表，是由尚未平仓的成交按照成交先后组成的链表 平仓时总是从链表头上开始逐一配对平仓，即先开先平原则
YDPositionForCommission	当前主键下的用于手续费计算的昨持仓量，仅在不区分今仓和昨仓时有效，此时用于手续费计算的今持仓量为Position-YDPositionForCommission

组合持仓是已经参与组合的持仓链表，按照组合的先后顺序依次排序；单腿持仓是尚未参与组合的持仓链表，按照成交先后顺序依次排序。由于组合持仓明细的结构变化较为频繁，我们并没有在上述持仓结构中直接开放组合持仓和单腿持仓的入口，如果需要查询组合持仓请参考相关查询函数。目前只有不区分今仓和昨仓的交易所支持组合和解组。

- 当交易所区分今仓和昨仓时，
  - 开仓时会对PositionDate为YD_PSD_Today的持仓完成以下操作：
    - 增加对应持仓的Position以更新总持仓数量
    - 将新开仓成交添加到上述持仓的用于盈亏计算的持仓链表的队尾
  - 平仓时投资者需要指定YDInputOrder.OffsetFlag为YD_OF_CloseToday或者YD_OF_CloseYesterday分别表示平今或者平昨（如果设置为YD_OF_Close则会被自动转换为YD_OF_CloseYesterday），会对受影响的持仓完成以下操作：
    - 扣减今仓或者昨仓的Position以更新今仓和昨仓的数量
    - 按照先开先平的顺序更新盈亏计算的持仓链表
- 当交易所不区分今仓和昨仓时，
  - 开仓时会对PositionDate为YD_PSD_Yesterday的持仓完成以下操作：
    - 增加对应持仓的Position以更新总持仓数量
    - 将新开仓成交添加到上述持仓的用于盈亏计算的持仓链表的队尾
    - 若交易所支持组合持仓业务，将新开仓成交添加到上述持仓的组合持仓链表的队尾
  - 平仓时投资者需要指定YDInputOrder.OffsetFlag为YD_OF_Close表示平仓（如果设置为YD_OF_CloseToday或者YD_OF_CloseYesterday则会被自动转换为YD_OF_Close），会对PositionDate为YD_PSD_Yesterday的持仓完成以下操作：
    - 扣减对应持仓的Position以更新总持仓数量
    - 按照先开先平的顺序更新盈亏计算的持仓链表
    - 若YDExchange.CloseTodayFirst为false，则优先扣减YDPositionForCommission；若YDExchange.CloseTodayFirst为true，只有当“平仓量>用于手续费计算的今持仓量”时，才需要从YDPositionForCommission中扣减用于手续费计算的今持仓量无法覆盖的部分

        - 若交易所支持组合持仓自动解组合业务，则优先按照先开先平的顺序匹配单腿持仓中的项直到覆盖平仓量；若无法完全覆盖的，则继续按照先开先平的顺序匹配组合持仓中的项直到覆盖剩余平仓量，并且修改涉及到的组合持仓中对方腿的组合持仓以及单腿持仓
    - 组合时，从待组合的两个持仓的单腿持仓的头部开始取出能覆盖组合数量的项，并添加到两个持仓的组合持仓的末尾；解组合时，从待组合的两个持仓的组合持仓链表里取出能覆盖解组合数量的项，同时插入到两个持仓的单腿持仓的合适位置，即能保持单腿持仓中的项按照成交先后顺序排序

5.2. 查询持仓

本节内容主要介绍ydExtendedApi中查询实时持仓的方法，所有查询都是基于ydExtendedApi的本地数据执行的，不会到柜台端去执行查询操作，由于所有的查询操作都有加锁操作，性能上会有一定损失。所有查询方法都必须在notifyCaughtUp以后调用，否则会因为报单成交数据不完整而导致持仓数据不准确。要查询日初持仓请参考日初持仓和日初组合持仓。

5.2.1. 查询期货与期权持仓

要获取单个期货与期权持仓可以调用getExtendedPosition方法，返回的YDExtendedPosition结构请参考持仓结构。

```
virtual const YDExtendedPosition *getExtendedPosition(int positionDate,int positionDirection,int hedgeFlag,
    const YDInstrument *pInstrument,const YDAccount *pAccount=NULL,bool create=false)=0;
```

调用上述方法的参数说明如下：

参数	说明
positionDate	要查询的持仓的持仓日期 YD_PSD_History：昨仓 YD_PSD_Today：今仓 若交易所不区分今仓和昨仓，请使用YD_PSD_History
positionDirection	要查询的持仓的持仓方向 YD_PD_Long：多头持仓 YD_PD_Short：空头持仓

参数	说明
hedgeFlag	要查询的持仓的投保标志，期货交易所与股票期权交易所的定义不同。 期货交易所如下： YD_HF_Speculation=1：投机 YD_HF_Arbitrage=2：套利，只有中金所支持。为便于编写程序，易达为是否支持套利交易和持仓提供了参数化表示，YDExchange.UseArbitragePosition为true时表示该交易所支持套利交易和持仓，否则为不支持 YD_HF_Hedge=3：套保 股票期权交易所如下： YD_HF_Normal=1：普通 YD_HF_Covered=3：备兑
pInstrument	要查询的合约的指针
pAccount	填写NULL，表示使用当前API登陆账户
create	当找不到持仓时是否创建空持仓。 若为true，当找不到持仓时会返回一条新初始化的持仓量为0的持仓；若为false，当找不到持仓时会返回NULL

易达提供了上述两种不同的批量查询持仓的方法，其具有相同的查询参数：

```
/// positions must have spaces of count, return real number of positions(may be greater than count). Only partial will be set if no enough space
virtual unsigned findExtendedPositions(const YDExtendedPositionFilter *pFilter,unsigned count,const YDExtendedPosition *positions[])=0;

/// User should call destroy method of return object to free memory after using following method
virtual YDQueryResult<YDExtendedPosition> *findExtendedPositions(const YDExtendedPositionFilter *pFilter)=0;
```

判断某持仓是否符合查询条件的伪代码逻辑如下所示：

```
if YDExtendedPositionFilter.PositionDate>=0 and YDExtendedPosition.PositionDate!=YDExtendedPositionFilter.PositionDate:
  return false;

if YDExtendedPositionFilter.PositionDirection>=0 and YDExtendedPosition.PositionDirection!=YDExtendedPositionFilter.PositionDirection:
  return false;

if YDExtendedPositionFilter.HedgeFlag>=0 and YDExtendedPosition.HedgeFlag!=YDExtendedPositionFilter.HedgeFlag:
```

```
  return false;

if YDExtendedPositionFilter.pInstrument!=NULL and
YDExtendedPosition.Instrument!=YDExtendedPositionFilter.pInstrument:
  return false;

if YDExtendedPositionFilter.pProduct!=NULL and
YDExtendedPosition.Product!=YDExtendedPositionFilter.pProduct:
  return false;

if YDExtendedPositionFilter.pExchange!=NULL and
YDExtendedPosition.Exchange!=YDExtendedPositionFilter.pExchange:
  return false;

return true;
```

参数YDExtendedPositionFilter各个字段填写说明如下：

字段	说明
PositionDate	要查询的持仓的持仓日期，设置为0表示该参数不生效 YD_PSD_History：昨仓 YD_PSD_Today：今仓
PositionDirection	要查询的持仓的持仓方向，设置为0表示该参数不生效 YD_PD_Long：多头持仓 YD_PD_Short：空头持仓
HedgeFlag	要查询的持仓的投保标志，期货交易所与股票期权交易所的定义不同。设置为0表示该参数不生效 期货交易所如下： YD_HF_Speculation=1：投机 YD_HF_Arbitrage=2：套利，只有中金所支持。为便于编写程序，易达为是否支持套利交易和持仓提供了参数化表示，YDExchange.UseArbitragePosition为true时表示该交易所支持套利交易和持仓，否则为不支持 YD_HF_Hedge=3：套保 股票期权交易所如下： YD_HF_Normal=1：普通 YD_HF_Covered=3：备兑
pInstrument	合约指针，设置为NULL则不限制
pProduct	产品指针，设置为NULL则不限制
pExchange	交易所指针，设置为NULL则不限制
pAccount	投资者请始终设置为NULL

第一种方法需要投资者事先分配固定长度的YDExtendedPosition指针数组，如果预分配的长度不足以容纳的，只会填入预分配数组长度的持仓。无论预分配长度是否足够，本方法的返回值都是返回满足查询条件的持仓的总数，投资者可以利用这个特点，可以调用findExtendedPositions(pFilter, 0, NULL)来快速获取满足条件的持仓的总数。

- 本方法的好处是在持仓不多而且预分配数组长度足够时，可以重复利用预分配的指针数组，而不用每次查询都分配
- 本方法的缺点是如果持仓较多时，可能需要两次调用（第一次获取总持仓数，第二次分配可容纳所有持仓的数组后再次调用）才能完整获取所有满足条件的持仓

第二种方法可以帮助投资者分配能容纳所有满足条件的持仓的空间，但是需要投资者在使用完成后主动调用YDQueryResult.destory()销毁分配的空间，不可使用delete删除。相比起第一种方法：

- 本方法的好处是调用一次就返回所有的持仓
- 本方法的缺点是需要投资者主动释放由本方法分配的空间，而且每次调用都会分配新的空间，频繁的分配与释放对缓存非常不友好。

5.2.2. 查询现货持仓

要获取单个现货持仓可以调用getExtendedSpotPosition方法。目前仅用于查询股票期权的基础合约ETF的持仓，查询结果可能不正确，请参考同步现货持仓。

```
virtual const YDExtendedSpotPosition *getExtendedSpotPosition(const YDInstrument *pInstrument,const YDAccount *pAccount=NULL,bool create=false)
```

调用上述方法的参数说明如下：

参数	说明
pInstrument	要查询的合约的指针
pAccount	填写NULL，表示使用当前API登陆账户
create	当找不到持仓时是否创建空持仓。 若为true，当找不到持仓时会返回一条新初始化的持仓量为0的持仓；若为false，当找不到持仓时会返回NULL

返回的YDExtendedSpotPosition结构体如下所示：

字段	说明
Position	总持仓量
ExchangeFrozenVolume	用于备兑开仓的锁定持仓量
CoveredVolume	备兑开仓占用量
ExecVolume	E日看跌期权行权锁定数量

字段	说明
ExecAllocatedVolume	E+1日行权分配量，E+1日盘中不会变化
ExecAllocatedAmount	E+1日行权分配金额，E+1日盘中不会变化。计算方式请参考日初持仓。

易达提供了批量查询持仓的方法，本方法可以帮助投资者分配能容纳所有满足条件的持仓的空间，但是需要投资者在使用完成后主动调用YDQueryResult.destory()销毁分配的空间。

```
/// User should call destroy method of return object to free memory after using following method
virtual YDQueryResult<YDExtendedSpotPosition> *findExtendedSpotPositions(const YDExtendedSpotPositionFilter *pFilter)
```

判断某持仓是否符合查询条件的伪代码逻辑如下所示：

```
if YDExtendedSpotPositionFilter.pInstrument!=NULL and YDExtendedSpotPosition.Instrument!=YDExtendedSpotPositionFilter.pInstrument:
  return false;

if YDExtendedSpotPositionFilter.pProduct!=NULL and YDExtendedSpotPosition.Product!=YDExtendedSpotPositionFilter.pProduct:
  return false;

if YDExtendedSpotPositionFilter.pExchange!=NULL and YDExtendedSpotPosition.Exchange!=YDExtendedSpotPositionFilter.pExchange:
  return false;

return true;
```

参数YDExtendedSpotPositionFilter各个字段填写说明如下：

字段	说明
pInstrument	合约指针，对于非YD_YOF_CombPosition的报单生效。如设置为NULL则不限制。
pProduct	产品指针，对于非YD_YOF_CombPosition的报单生效。如设置为NULL则不限制。
pExchange	交易所指针。如设置为NULL则不限制。
pAccount	投资者请始终设置为NULL

5.2.2.1. 同步现货持仓

目前易达没有现货柜台，无法连接到交易所查询现货持仓情况，从而盘中易达中的现货持仓始终是日初持仓。由于此时现货持仓大概率是不准确的，柜台不会检查锁仓、备兑开仓、看跌期权行权的现货持仓量，这就会造成以下问题：

- 锁仓时若实际可锁仓位不足的，交易所会返回锁定错误
- 备兑开仓时若实际可用现货仓位不足的，交易所会返回开仓错误
- 看跌期权行权时实际可用现货仓位不足的，**交易所不会检查现货仓位且会接受行权指令**，这会导致投资者误以为现货仓位足够。但实际上结算时该行权指令会失败，从而使得投资者蒙受损失

为了能控制风险以及获取准确的现货持仓，易达提供了现货柜台连接器，该连接器会定时从现货柜台（目前只支持CTP）查询现货持仓并同步到易达股票期权柜台。若经纪商启用了现货柜台连接器，API会通过以下回调函数通知现货持仓同步处于激活状态。该通知消息是持续发送的，若30秒内没有收到通知，则表示同步机制中断，请及时联系经纪商排查问题。

```
virtual void notifySpotAlive(const YDExchange *pExchange)
```

在同步激活状态下，若现货柜台上的持仓发生变化，ydApi和ydExtendedApi会通过以下回调收到通知，newPosition为更新后的现货持仓量：

```
virtual void notifySpotPosition(const YDInstrument *pInstrument,const
YDAccount *pAccount,int newPosition)
```

若投资者使用了YDExtendedListener，那么在任何notifySpotPosition回调的时候都会同时回调notifyExtendedSpotPosition：

```
virtual void notifyExtendedSpotPosition(const YDExtendedSpotPosition
*pSpotPosition)
```

5.2.3. 查询组合持仓明细

易达为查询上交所和深交所的组合持仓明细提供了以下方法，其中combPositionDetailID是唯一确定组合明细的主键：

```
/// getCombPositionDetail can only be used in SSE/SZSE
virtual const YDExtendedCombPositionDetail *getCombPositionDetail(int
combPositionDetailID)=0;
```

返回的YDExtendedCombPositionDetail结构体的字段如下所示：

字段	说明
m_pAccount	组合持仓明细属于的投资者的指针
m_pCombPositionDef	组合持仓定义指针

字段	说明
Position	组合持仓明细量
CombPositionDetailID	组合持仓明细ID，只有当在上交所和深交所时有效

易达提供了上述两种不同的批量查询组合持仓明细的方法，其具有相同的查询参数：

```
/// combPositionDetails must have spaces of count, return real number of combPositionDetails(may be greater than count). Only partial will be set if no enough space
virtual unsigned findCombPositionDetails(const YDCombPositionDetailFilter *pFilter,unsigned count,const YDExtendedCombPositionDetail *combPositionDetails[])=0;

/// User should call destroy method of return object to free memory after using following method
virtual YDQueryResult<YDExtendedCombPositionDetail> *findCombPositionDetails(const YDCombPositionDetailFilter *pFilter)=0;
```

判断某组合持仓明细是否符合查询条件的伪代码逻辑如下所示，为便于阅读将YDExtendedCombPositionDetail简写为Detail，YDCombPositionDetailFilter简写为Filter：

```
if Filter.IncludeSplit==false and Detail.Position<=0:
  return false

if Filter.pAccount!=NULL and Detail.m_pAccount!=Filter.pAccount:
  return false

if Filter.pCombPositionDef!=NULL and Detail.m_pCombPositionDef!=Filter.pCombPositionDef:
  return false

if Filter.pInstrument!=NULL:
  hasMatchLeg=false
  for (int legID=0;legID<2;legID++):
    if matchLeg(Detail->m_pCombPositionDef,legID,Filter):
      hasMatchLeg=true
      break
  if hasMatchLeg==false:
    return false

return true

bool matchLeg(YDCombPositionDef Def,int legID,YDCombPositionDetailFilter Filter):
  if Def.m_pInstrument[legID]!=Filter.pInstrument:
```

```
23      return false
24
25    if Filter.PositionDirection!=0 and
Filter.PositionDirection!=Def.PositionDirection[legID]:
26      return false
27
28    return true
```

参数YDExtendedPositionFilter各个字段填写说明如下：

字段	说明
pCombPositionDef	组合持仓定义指针，设置为NULL则不限制
pInstrument	合约指针，设置为NULL则不限制
PositionDirection	要查询的组合持仓明细的持仓方向，设置为0表示该参数不生效 YD_PD_Long：多头持仓 YD_PD_Short：空头持仓
IncludeSplit	是否包含已经解组的组合持仓明细

第一种方法需要投资者事先分配固定长度的YDExtendedCombPositionDetail指针数组，如果预分配的长度不足容纳的，只会填入预分配数组长度的组合持仓明细。无论预分配长度是否足够，本方法的返回值都是返回满足查询条件的组合持仓明细的总数，投资者可以利用这个特点，可以调用findCombPositionDetails(pFilter, 0, NULL)来快速获取满足条件的组合持仓明细的总数。

- 本方法的好处是在组合持仓明细不多而且预分配数组长度足够时，可以重复利用预分配的指针数组，而不用每次查询都分配
- 本方法的缺点是如果组合持仓明细较多时，可能需要两次调用（第一次获取总组合持仓明细数，第二次分配可容纳所有组合持仓明细的数组后再次调用）才能完整获取所有满足条件的组合持仓明细

第二种方法可以帮助投资者分配能容纳所有满足条件的组合持仓明细的空间，但是需要投资者在使用完成后主动调用YDQueryResult.destory()销毁分配的空间。相比起第一种方法：

- 本方法的好处是调用一次就返回所有的组合持仓明细
- 本方法的缺点是需要投资者主动释放由本方法分配的空间，而且每次调用都会分配新的空间，频繁的分配与释放对缓存非常不友好

5.3. 保证金

易达的期货交易所保证金收取和优惠原则是尽可能**与主流主席柜台系统保持一致**。需要特别说明的是，易达和主流主席柜台系统一样对上期所、能源所和中金所的挂单与持仓一起合并计算大边保证金，但是目前在以下两点和主流主席柜台系统在业务规则上不一致：

- 大商所期权组合保证金，易达与主流主席柜台系统的计算方法不一样
- 郑商所对锁单，易达把持仓和挂单一起按照大边计算优惠了保证金，主流主席柜台系统目前只计算了持仓大边优惠而没有计算挂单的大边优惠。

因此，除了上述两个业务例外之外，在交易所休市阶段易达和主流主席柜台系统 的保证金授权应该是完全一致的，在有交易的时候，会因为交易所回报顺序、行情更新时机、柜台重算时机不同，而导致易达柜台和主流主席柜台系统有细微的差异。

易达并不区分报单冻结保证金和持仓保证金，而是统一当做保证金处理，计算大边保证金时也是把报单保证金和持仓保证金统一加总后计算大边的。本文中我们为了方便投资者理解，我们区分了报单保证金和持仓保证金，但是请投资者注意在柜台内部是不区分的。

5.3.1. 保证金率

要计算持仓的保证金，首先要获取该合约在对应的投保标志上的保证金率，我们建议投资者直接使用getInstrumentMarginRate获取对应的保证金率，而不要直接从YDAccountInstrumentInfo中读取。

```
virtual const YDMarginRate *getInstrumentMarginRate(const YDInstrument
*pInstrument,int hedgeFlag,const YDAccount *pAccount=NULL)
```

返回的YDMarginRate结构体使用了大量的union字段，其目的是为了适应不同的保证金模型，为了方便理解，我们将分别列出三种不同保证金模型下的保证金率参数。

以下字段适用于期货交易所的期货保证金。

字段	说明
m_pAccount	保证金率所属投资者的指针
m_pProduct	保证金率所属产品的指针
m_pInstrument	保证金率对应的合约的指针
HedgeFlag	YD_HF_Speculation=1：投机 YD_HF_Arbitrage=2：套利，只有中金所支持。为便于编写程序，易达为是否支持套利交易和持仓提供了参数化表示，YDExchange.UseArbitragePosition为true时表示该交易所支持套利交易和持仓，否则为不支持 YD_HF_Hedge=3：套保
LongMarginRatioByMoney	基于金额的多头保证金率
LongMarginRatioByVolume	基于手数的多头保证金率
ShortMarginRatioByMoney	基于金额的空头保证金率
ShortMarginRatioByVolume	基于手数的空头保证金率

以下字段适用于期货交易所的商品期权保证金和中金所的股指期权保证金。

字段	说明
m_pAccount	保证金率所属投资者的指针

字段	说明
m_pProduct	保证金率所属产品的指针
m_pInstrument	保证金率对应的合约的指针
HedgeFlag	YD_HF_Speculation=1：投机 YD_HF_Arbitrage=2：套利，只有中金所支持。为便于编写程序，易达为是否支持套利交易和持仓提供了参数化表示，YDExchange.UseArbitragePosition为true时表示该交易所支持套利交易和持仓，否则为不支持 YD_HF_Hedge=3：套保
CallMarginRatioByMoney	基于金额的看涨期权空头保证金率
CallMarginRatioByVolume	基于手数的看涨期权空头保证金率
PutMarginRatioByMoney	基于金额的看跌期权空头保证金率
PutMarginRatioByVolume	基于手数的看跌期权空头保证金率

以下字段适用于上交所和深交所的股票期权保证金。

字段	说明
m_pAccount	保证金率所属投资者的指针
m_pProduct	保证金率所属产品的指针
m_pInstrument	保证金率对应的合约的指针
HedgeFlag	YD_HF_Normal=1：普通 YD_HF_Covered=3：备兑
BaseMarginRate	基础合约保证金率
LinearFactor	线性系数
LowerBoundaryCoef	最低保障系数

5.3.1.1. 盘中调保

通常情况下，保证金率在盘中交易时是固定不变了，但在某些情况下经纪商可能会在盘中对某些客户调整保证金。当经纪商更新盘中保证金率时，投资者会收到以下回调：

```
virtual void notifyUpdateMarginRate(const YDUpdateMarginRate
*pUpdateMarginRate)
```

当保证金率发生改变时，使用ydExtendedApi的投资者不用关心究竟是哪些合约发生了改变以及需要更新哪些持仓的保证金，ydExtendedApi会妥善管理好保证金率变更的相关工作，但是使用ydApi的投资者就需要自主更新保证金率发生变化的合约对应的持仓保证金。下表为更新保证金率YDUpdateMarginRate中表示其影响保证金率的合约过滤字段，保证金率相关字段不予列出，需要请参考上文各个保证金模型对应的保证金率说明。

字段	说明
AccountRef	账户序号
ProductRef	产品序号
InstrumentRef	合约序号
HedgeFlagSet	投保标志集合，若某投保标志受到影响，1<<HedgeFlag对应的位会被设置
OptionTypeSet	期权类型集合，若某类型期权受到影响，1<<OptionsType对应的位会被设置
ExpireDate	合约到期日，主要用于表示期权系列
UnderlyingInstrumentRef	基础合约序号
Multiple	合约乘数，股票期权可能会受到分红影响而调整其保证金率

下面我们给出了当收到某次保证金率变更时，判断该变更是否会影响到某持仓（合约与投保标志）的示例代码：

```
bool applyToInstrument(const CYDUpdateMarginRate *pUpdateMarginRate,const YDInstrument *pInstrument, int hedgeFlag) const
{
  if ((pUpdateMarginRate->ProductRef>=0) && (pInstrument->ProductRef!=pUpdateMarginRate->ProductRef))
  {
    return false;
  }
  if ((pUpdateMarginRate->InstrumentRef>=0) && (pInstrument->InstrumentRef!=pUpdateMarginRate->InstrumentRef))
  {
    return false;
  }
  if ((pUpdateMarginRate->UnderlyingInstrumentRef>=0) && (pInstrument->UnderlyingInstrumentRef!=pUpdateMarginRate->UnderlyingInstrumentRef))
  {
    return false;
  }
  if ((pUpdateMarginRate->ExpireDate>0) && (pInstrument->ExpireDate!=pUpdateMarginRate->ExpireDate))
```

```
  {
    return false;
  }
  if ((pUpdateMarginRate->Multiple>0) && (pInstrument->Multiple!=pUpdateMarginRate->Multiple))
  {
    return false;
  }
  if ((pUpdateMarginRate->OptionTypeSet>0) && (((1<<pInstrument->OptionsType)&pUpdateMarginRate->OptionTypeSet)==0))
  {
    return false;
  }
  if ((pUpdateMarginRate->HedgeFlagSet>0) && (((1<<hedgeFlag)&pUpdateMarginRate->HedgeFlagSet)==0))
  {
    return false;
  }
  return true;
}
```

我们建议ydApi的用户遍历持仓刷新各个持仓的保证金，伪代码如下：

```
virtual void notifyUpdateMarginRate(const YDUpdateMarginRate *pUpdateMarginRate)
{
  for(auto pPosition : AllPositions)
  {
    if (applyToInstrument(pUpdateMarginRate, pPosition->pInstrument, pPosition->hedgeFlag))
    {
      // 使用投资者自己的保证金刷新方法
      userDefinedUpdateMargin(pPosition, pUpdateMarginRate);
    }
  }
}
```

5.3.2. 期货保证金

期货多头保证金计算公式为：

$$\text{期货多头持仓保证金} = (\text{价格} \times \text{合约乘数} \times \text{基于金额的多头保证金率} + \text{基于手数的多头保证金率}) \times \text{数量}$$

期货空头保证金计算公式为：

$$\text{期货空头持仓保证金} = (\text{价格} \times \text{合约乘数} \times \text{基于金额的空头保证金率} + \text{基于手数的空头保证金率}) \times \text{数量}$$

对于**报单**，计算期货报单保证金时使用的数量为报单量，价格受到YDSystemParam中（Name, Target）为（OrderMarginBasePrice, Futures）的参数值控制，以下列举不同参数值时系统的处理方式：

- YD_CBT_PreSettlementPrice：昨结算价
- YD_CBT_OrderPrice：当报单为市价单时，价格为涨停板价，当报单为限价单/FAK/FOK时，价格为报单价格。这是默认配置的方式，也是主流主席柜台系统当前使用的方式
- YD_CBT_SamePrice：与期货持仓保证金率采用相同的价格，即使用YDSystemParam中（Name, Target）为（MarginBasePrice, Futures）的参数值。若此时期货持仓保证金率使用的是YD_CBT_OpenPrice时，期货报单保证金使用的价格是昨结算价

对于**昨仓**，计算期货持仓保证金时使用的数量为持仓量，价格始终使用昨结算价。此处的昨仓是指按照逐日盯市规则结算后剩余的持仓，并非持仓结构中今昨仓的概念。

对于**今仓**，计算期货持仓保证金时使用的数量为持仓量，价格受到YDSystemParam中（Name, Target）为（MarginBasePrice, Futures）的参数值控制，以下列举不同参数值时系统的处理方式：

- YD_CBT_PreSettlementPrice：昨结算价
- YD_CBT_OpenPrice：开仓价，按照开仓价逐个计算持仓明细对应的保证金。这是默认配置的方式，也是主流主席柜台系统当前使用的方式
- YD_CBT_LastPrice：最新价
- YD_CBT_MarketAveragePrice：市场平均成交价
- YD_CBT_MaxLastPreSettlementPrice：最新价和昨结算价的大者

5.3.3. 期权保证金

5.3.3.1. 商品期权保证金

适用于上期所、能源所、大商所、广期所、郑商所交易的商品期权。

看涨商品期权的保证金公式如下：

$$\text{商品期权看涨保证金} = [\text{期权价格} \times \text{期权合约乘数} + max(\text{看涨期权基础保证金}–\text{看涨期权虚值}/2, \text{看涨期权基础保证金}/2)] \times \text{数量}$$

$$\text{看涨期权基础保证金} = \text{基础合约价格} \times \text{基础合约乘数} \times \text{基于金额的看涨期权空头保证金率} + \text{基于手数的看涨期权空头保证金率}$$

$$\text{看涨期权虚值} = max((\text{行权价}–\text{基础合约价格}) \times \text{期权合约乘数}, 0)$$

看跌商品期权的保证金公式如下：

$$\text{商品期权看跌保证金} = [\text{期权价格} \times \text{期权合约乘数} + max(\text{看跌期权基础保证金}–\text{看涨期权虚值}/2, \text{看跌期权基础保证金}/2)] \times \text{数量}$$

$$\text{看跌期权基础保证金} = \text{基础合约价格} \times \text{基础合约乘数} \times \text{基于金额的看跌期权空头保证金率} + \text{基于手数的看跌期权空头保证金率}$$

$$\text{看跌期权虚值} = max((\text{基础合约价格}–\text{行权价}) \times \text{期权合约乘数}, 0)$$

对于**报单**，计算期权报单保证金时使用的数量为报单量，期权价格受到YDSystemParam中（Name, Target）为（OrderMarginBasePrice, Options）的参数值控制，以下列举不同参数值时系统的处理方式：

- YD_CBT_PreSettlementPrice：昨结算价。这是默认配置的方式，也是主流主席柜台系统当前使用的方式
- YD_CBT_OrderPrice：当报单为市价单时，价格为涨停板价，当报单为限价单/FAK/FOK时，价格为报单价格。
- YD_CBT_SamePrice：与期权持仓保证金率采用相同的价格，即使用YDSystemParam中（Name, Target）为（MarginBasePrice, Options）的参数值。若此时期权持仓保证金率使用的是YD_CBT_OpenPrice时，期权报单保证金使用的价格是昨结算价

对于**昨仓**和**今仓**，计算期权持仓保证金时使用的数量为持仓量，期权价格受到YDSystemParam中（Name, Target）为（MarginBasePrice, Options）的参数值控制，以下列举不同参数值时系统的处理方式：

- YD_CBT_PreSettlementPrice：昨结算价
- YD_CBT_OpenPrice：开仓价，按照开仓价逐个计算持仓明细对应的保证金
- YD_CBT_LastPrice：最新价
- YD_CBT_MarketAveragePrice：市场平均成交价
- YD_CBT_MaxLastPreSettlementPrice：最新价和昨结算价的大者。这是默认配置的方式，也是主流主席柜台系统当前使用的方式

对于**报单**、**昨仓**和**今仓**，基础合约价格受到YDSystemParam中（Name, Target）为（MarginBasePriceAsUnderlying,Options）的参数值控制，以下列举不同参数值时系统的处理方式：

- YD_CBT_PreSettlementPrice：昨结算价。这是默认配置的方式，也是主流主席柜台系统当前使用的方式
- YD_CBT_LastPrice：最新价
- YD_CBT_MaxLastPreSettlementPrice：最新价和昨结算价的大者

5.3.3.2. 股指期权保证金

适用于中金所交易的股指期权。

股指期权看涨保证金的计算公式为：

$$\text{股指期权看涨保证金} = (\text{期权价格} \times \text{期权合约乘数} + max(\text{看涨期权基础保证金} - \text{看涨期权虚值}, \text{最低保障系数} \times \text{看涨期权基础保证金})) \times \text{数量}$$

$$\text{看涨期权基础保证金} = \text{基础合约价格} \times \text{基础合约乘数} \times \text{基于金额的看涨期权空头保证金率}$$

$$\text{看涨期权虚值} = max((\text{行权价} - \text{基础合约价格}) \times \text{期权合约乘数}, 0)$$

股指期权看跌保证金的计算公式为：

$$\text{股指期权看跌保证金} = (\text{期权价格} \times \text{期权合约乘数} + max(\text{看跌期权基础保证金}1 - \text{看跌期权虚值}, \text{最低保障系数} \times \text{看跌期权基础保证金}2)) \times \text{数量}$$

$$\text{看跌期权基础保证金}1 = \text{基础合约价格} \times \text{基础合约乘数} \times \text{基于金额的看跌期权空头保证金率}$$

$$\text{看跌期权基础保证金}2 = \text{行权价} \times \text{基础合约乘数} \times \text{基于金额的看跌期权空头保证金率}$$

$$\text{看跌期权虚值} = max\left(\left(\text{基础合约价格} - \text{行权价}\right) \times \text{期权合约乘数}, 0\right)$$

最低保障系数取自于YDSystemParam中（Name, Target）为（MarginLowerBoundaryCoef,MarginCalcMethod1）的参数值，目前股指期权的最低保障系数应该为0.5。

对于**报单**，计算期权报单保证金时使用的数量为报单量，期权价格受到YDSystemParam中（Name, Target）为（OrderMarginBasePrice, Options）的参数值控制，以下列举不同参数值时系统的处理方式：

- YD_CBT_PreSettlementPrice：昨结算价。这是默认配置的方式，也是主流主席柜台系统当前使用的方式
- YD_CBT_OrderPrice：当报单为市价单时，价格为涨停板价，当报单为限价单/FAK/FOK时，价格为报单价格。
- YD_CBT_SamePrice：与期权持仓保证金率采用相同的价格，即使用YDSystemParam中（Name, Target）为（MarginBasePrice, Options）的参数值。若此时期权持仓保证金率使用的是YD_CBT_OpenPrice时，期权报单保证金使用的价格是昨结算价

对于**昨仓**和**今仓**，计算期权持仓保证金时使用的数量为持仓量，期权价格受到YDSystemParam中（Name, Target）为（MarginBasePrice, Options）的参数值控制，以下列举不同参数值时系统的处理方式：

- YD_CBT_PreSettlementPrice：昨结算价
- YD_CBT_OpenPrice：开仓价，按照开仓价逐个计算持仓明细对应的保证金
- YD_CBT_LastPrice：最新价
- YD_CBT_MarketAveragePrice：市场平均成交价
- YD_CBT_MaxLastPreSettlementPrice：最新价和昨结算价的大者。这是默认配置的方式，也是主流主席柜台系统当前使用的方式

对于**报单**、**昨仓**和**今仓**，由于易达柜台盘中不获取股指行情，所以无论如何设置，股指期权的基础合约价格总是用昨结算价。

5.3.3.3. 股票期权保证金

在生产实践中，股票期权的保证金模型可以分为线性和非线性两种，易达柜台通过控制计算公式中的参数同时实现了这两个保证金模型。例如，当线性系数为1.2，其他参数为交易所标准值时（目前基础合约保证金率为12%，最低保障系数为7%），即采用线性模式；当基础合约保证金率为15%，最低保障系数为8%，而线性系数为1时，即采用非线性模式。

股票期权看涨保证金的计算公式为：

$$\begin{aligned}\text{股票期权看涨保证金} = \text{线性系数} \times \big(&\text{期权价格} + max\big(\text{基础合约保证金率} \times \text{基础合约价格} \\ &-\text{看涨虚值度}, \text{最低保障系数} \times \text{基础合约价格}\big)\big) \times \text{期权合约乘数} \times \text{数量}\end{aligned}$$

$$\text{看涨虚值度} = max\left(\text{行权价} - \text{基础合约价格}, 0\right)$$

股票期权看跌保证金的计算公式为：

$$\text{看跌期权空方的每手保证金} = \text{线性系数} \times (\text{期权价格} + min(max(\text{基础合约保证金率} \times \text{基础合约价格} - \text{看跌虚值度}, \text{最低保障系数} \times \text{行权价}), \text{行权价}-\text{期权最新价})) \times \text{期权合约乘数} \times \text{数量}$$

$$\text{看跌虚值度} = max(\text{基础合约价格} - \text{行权价}, 0)$$

对于**报单**，计算期权报单保证金时使用的数量为报单量，期权价格受到YDSystemParam中（Name, Target）为（OrderMarginBasePrice, Options）的参数值控制，以下列举不同参数值时系统的处理方式：

- YD_CBT_PreSettlementPrice：昨结算价。这是默认配置的方式，也是主流主席柜台系统当前使用的方式
- YD_CBT_OrderPrice：当报单为市价单时，价格为涨停板价，当报单为限价单/FAK/FOK时，价格为报单价格。
- YD_CBT_SamePrice：与期权持仓保证金率采用相同的价格，即使用YDSystemParam中（Name, Target）为（MarginBasePrice, Options）的参数值。若此时期权持仓保证金率使用的是YD_CBT_OpenPrice时，期权报单保证金使用的价格是昨结算价

对于**昨仓**和**今仓**，计算期权持仓保证金时使用的数量为持仓量，期权价格受到YDSystemParam中（Name, Target）为（MarginBasePrice, Options）的参数值控制，以下列举不同参数值时系统的处理方式：

- YD_CBT_PreSettlementPrice：昨结算价
- YD_CBT_OpenPrice：开仓价，按照开仓价逐个计算持仓明细对应的保证金
- YD_CBT_LastPrice：最新价
- YD_CBT_MarketAveragePrice：市场平均成交价
- YD_CBT_MaxLastPreSettlementPrice：最新价和昨结算价的大者。这是默认配置的方式，也是主流主席柜台系统当前使用的方式

对于**报单**、**昨仓**和**今仓**，基础合约价格受到YDSystemParam中（Name, Target）为（MarginBasePriceAsUnderlying,Options）的参数值控制，以下列举不同参数值时系统的处理方式：

- YD_CBT_PreSettlementPrice：昨结算价。这是默认配置的方式，也是主流主席柜台系统当前使用的方式
- YD_CBT_LastPrice：最新价
- YD_CBT_MaxLastPreSettlementPrice：最新价和昨结算价的大者

5.3.4. 期权行权保证金

5.3.4.1. 商品期权行权保证金

商品期权的看涨期权保证金的计算公式为：

$$\text{商品期权看涨行权保证金} = ((\text{基础合约昨结算价} \times \text{基础合约乘数} \times \text{基础合约的基于金额的多头保证金率} + \text{基础合约的基于手数的多头保证金率}) \times \text{期权基础乘数} + max(\text{行权价} - \text{基础合约昨结算价}, 0) \times \text{期权合约乘数}) \times \text{数量}$$

商品期权的看跌期权保证金的计算公式为：

$$\begin{aligned}\text{商品期权看跌行权保证金} = (&(\text{基础合约昨结算价} \times \text{基础合约乘数} \times \text{基础合约的基于金额的空头保证金率} \\ &+\text{基础合约的基于手数的空头保证金率}) \times \text{期权基础乘数} \\ &+max(\text{基础合约昨结算价} - \text{行权价}, 0) \times \text{期权合约乘数}) \times \text{数量}\end{aligned}$$

基础乘数为期权合约的UnderlyingMultiply。

5.3.4.2. 股票期权行权保证金

股票期权看涨行权保证金的计算公式为：

$$\text{股票期权看涨行权保证金} = \text{行权价} \times \text{期权乘数}$$

股票期权看跌行权不收取保证金。

5.3.5. 保证金优惠

交易所优惠保证金的主要手段分为单向大边保证金和组合保证金两种，虽然最终的效果类似，但是其使用方式和复杂程度截然不同，请投资者区别看待两种优惠方式，不可混为一谈。目前中金所、上期所、能源所、郑商所支持大边保证金业务，大商所、郑商所、上交所和深交所支持组合保证金业务。

5.3.5.1. 单向大边保证金

郑商所支持**同合约单向大边保证金**，具体方法为分别将属于同一合约的多头和空头的投机、套保仓位和报单的保证金加总后，取保证金较大的一边为该合约上实际收取的保证金。

上期所和能源所支持**同产品单向大边保证金**，具体方法为分别将属于同一产品的多头和空头的投机、套保仓位和报单的保证金加总后，取保证金较大的一边为该产品上实际收取的保证金。

中金所支持**跨产品单向大边保证金**，具体方法为分别将属于同一产品组的多头和空头的投机、套保仓位和报单的保证金加总后，取保证金较大的一边为该产品组上实际收取的保证金。产品组的定义在YDProduct.m_pMarginProduct中，该字段指向了产品组的组长产品（组长产品并不是固定的，可能会随着初始化数据的变化而有所不同），组长产品和所有m_pMarginProduct指向该组长产品的属于同一产品组。目前中金所有两个产品组，所有的股指期货IF、IC、IH、IM属于同一产品组，所有的国债期货TF、TS、T属于同一产品组。

为方便策略程序识别需要参与单向大边保证金计算的合约，YDInstrument.SingleSideMargin为True时表示该合约需要参与单向大边保证金计算，否则不参与。通常情况下上期所、能源所和中金所的期货合约的SingleSideMargin为True，但由于临近交割的期货合约不应参与单向大边保证金计算，单向大边保证金计算中需要剔除这些合约，所以这部分临近交割的合约的SingleSideMargin会被设置为False。此部分设置来自于主流主席柜台系统的日初数据。

5.3.5.2. 组合保证金

易达为各交易所不同的组合类型制定了不同的组合保证金优惠方法，下面我们逐一介绍。

5.3.5.2.1. 期货组合

以下组合类型适用本优惠算法：

- YD_CPT_DCE_FuturesOffset：大商所期货对锁
- YD_CPT_DCE_FuturesCalendarSpread：大商所期货跨期
- YD_CPT_DCE_FuturesProductSpread：大商所期货跨品种
- YD_CPT_GFEX_FuturesOffset：广期所期货对锁
- YD_CPT_GFEX_FuturesCalendarSpread：广期所期货跨期
- YD_CPT_GFEX_FuturesProductSpread：广期所期货跨品种
- YD_CPT_CZCE_Spread：郑商所套利

期货组合保证金需要先按照规则选择小边，然后再使用选中小边持仓的保证金作为节约保证金，并从原有两腿的保证金之和中减去节约保证金，即得到组合保证金。小边的选择规则为：

- 比较两腿的交易所期货保证金，较小的一边被选中，若相同则继续。交易所期货保证金与期货保证金的计算公式相同，在计算时价格使用昨结算价，保证金率使用交易所保证金率，交易所保证金率可以在YDInstrument.m_pExchangeMarginRate中找到
- 若组合类型为YD_CPT_CZCE_Spread，则直接选择右腿，否则继续
- 比较两腿的交割月，较远月的一边被选中，若相同则继续
- 比较两腿的合约代码，较大的合约代码被选中。合约代码比较是字符串比较

5.3.5.2.2. 期权跨式

以下组合类型适用本优惠算法：

- YD_CPT_DCE_OptionsStraddle：大商所卖出期权跨式
- YD_CPT_DCE_OptionsStrangle：大商所卖出期权宽跨式
- YD_CPT_GFEX_OptionsStraddle：广期所卖出期权跨式
- YD_CPT_GFEX_OptionsStrangle：广期所卖出期权宽跨式
- YD_CPT_CZCE_StraddleStrangle：郑商所卖出跨式或宽跨式

上述期权组合保证金需要先按照规则选择小边，然后再使用选中小边计算节约保证金，并从原有两腿的保证金之和中减去节约保证金，即得到组合保证金。节约保证金的计算公式为：
$交易所期权保证金 \times 持仓量$，交易所期权保证金与期权保证金的计算公式相同，在计算时价格使用昨结算价，保证金率使用交易所保证金率，交易所保证金率可以在YDInstrument.m_pExchangeMarginRate中找到。小边的选择规则为：

- 比较两腿的期权保证金，较小的一边被选中，若相同则继续
- 计算两腿的节约保证金，选择较小的节约保证金为最终的节约保证金

以下组合类型适用本优惠算法：

- YD_CPT_StockOption_KS：上交所、深交所所跨式空头
- YD_CPT_StockOption_KKS：上交所、深交所宽跨式空头

上述期权组合保证金需要先按照规则选择小边，然后再使用选中小边计算节约保证金，并从原有两腿的保证金之和中减去节约保证金，即得到组合保证金。节约保证金的计算公式为：
$(期权保证金 - 期权昨结算价 \times 期权合约乘数 \times 线性系数) \times 持仓量$。小边的选择规则为：

- 比较两腿的期权保证金，较小的一边被选中，若相同则继续
- 计算两腿的节约保证金，选择较小的节约保证金为最终的节约保证金

5.3.5.2.3. 卖权覆盖

以下组合类型适用本优惠算法：

- YD_CPT_DCE_SellOptionsCovered：大商所卖出期权期货组合
- YD_CPT_GFEX_SellOptionsCovered：广期所卖出期权期货组合
- YD_CPT_CZCE_SellOptionConvered：郑商所卖出期权期货组合

直接使用左腿计算节约保证金，并从原有两腿的保证金之和中减去节约保证金，即得到组合保证金。节约保证金的计算公式为：$交易所期权保证金 \times 持仓量$。公式中的期权价格使用YDSystemParam中（Name, Target）为（MarginBasePrice, Options）的参数值指定的价格；交易所期权保证金与商品期权保证金的计算公式相同，在计算时价格使用昨结算价，保证金率使用交易所保证金率，交易所保证金率可以在YDInstrument.m_pExchangeMarginRate中找到。

5.3.5.2.4. 买权组合

以下组合类型适用本优惠算法：

- YD_CPT_DCE_OptionsOffset：大商所期权对锁
- YD_CPT_DCE_BuyOptionsVerticalSpread：大商所买入期权垂直套利
- YD_CPT_DCE_BuyOptionsCovered：大商所买入期权期货组合
- YD_CPT_GFEX_OptionsOffset：广期所期权对锁
- YD_CPT_GFEX_BuyOptionsVerticalSpread：广期所买入期权垂直套利
- YD_CPT_GFEX_BuyOptionsCovered：广期所买入期权期货组合

当投资者平上述组合的买腿时是需要加收保证金的，在主流主席柜台系统上若可用资金不足以加收保证金时会禁止平仓。我们认为禁止平仓会影响风险的释放，可能反而会造成更大的风险，同时也违反了易达平仓不检查资金的原则，因此在易达中平买腿时可能会导致可用资金变为负数。易达将选择权力交给经纪商，若经纪商对此处理有异议，可以关闭本组合保证金优惠功能，功能关闭后仍然可以进行组合但是不再优惠保证金，这样就确保平仓不会导致穿仓，若经纪商认可易达的处理方式，可以打开该功能。在初始安装中，该功能是默认关闭的。投资者可以通过查看YDSystemParam中（Name, Target）为（PortfolioMarginConcession, DCELongOptionPortfolio）的参数值了解该功能是否开启。

若开启了优惠，则直接使用右腿计算节约保证金，并从原有两腿的保证金之和中减去节约保证金，即得到组合保证金。节约保证金的计算公式为：$(1 - 组合保证金折扣) \times 期权保证金 \times 持仓量$。公式中的组合保证金折扣为YDCombPositionDef.Parameter，详情参见组合持仓定义。

5.3.5.2.5. 卖权垂直套利

以下组合类型适用本优惠算法：

- YD_CPT_DCE_SellOptionsVerticalSpread：大商所卖出期权垂直套利
- YD_CPT_GFEX_SellOptionsVerticalSpread：广期所卖出期权垂直套利

当投资者平上述组合的买腿时是需要加收保证金的，在主流主席柜台系统上若可用资金不足以加收保证金时会禁止平仓。我们认为禁止平仓会影响风险的释放，进而造成更大的风险，同时也违反了平仓不检查资金的通行做法，因此在易达中平买腿时可能会导致可用资金变为负数。易达将选择权交给经纪商：若经纪商对此处理有异议，可以关闭本组合保证金优惠功能，功能关闭后仍然可以进行组合但是不再优惠保证金，这样就确保平仓不会导致穿仓；若经纪商认可易达的处理方式，可以打开该功能。在初始安装中，该功能是默认关闭的。投资者可以通过查看YDSystemParam中（Name, Target）为（PortfolioMarginConcession, DCELongOptionPortfolio）的参数值了解该功能是否开启。

若开启了优惠，则直接使用右腿计算节约保证金，并从原有两腿的保证金之和中减去节约保证金，即得到组合保证金。节约保证金的计算公式为：

$$max(右腿期权保证金 - |左腿行权价 - 右腿行权价| \times 左腿合约乘数, 0) \times 持仓量$$

5.3.5.2.6. 牛熊市价差

以下组合类型适用下面优惠算法：

- YD_CPT_StockOption_CNSJC：上交所、深交所认购牛市价差
- YD_CPT_StockOption_PXSJC：上交所、深交所认沽熊市价差

直接使用右腿计算节约保证金，并从原有两腿的保证金之和中减去节约保证金，即得到组合保证金。节约保证金的计算公式为：$右腿每手保证金 \times 持仓量$。右腿每手保证金即为右腿实际的保证金，具体计算方法请参考股票期权保证金。

以下组合类型适用下面优惠算法：

- YD_CPT_StockOption_CXSJC：上交所、深交所认购熊市价差
- YD_CPT_StockOption_PNSJC：上交所、深交所认沽牛市价差

直接使用右腿计算节约保证金，并从原有两腿的保证金之和中减去节约保证金，即得到组合保证金。节约保证金的计算公式为：

$$(右腿每手保证金 - |左腿行权价 - 右腿行权价| \times 右腿合约乘数 \times 右腿线性系数) \times 组合持仓量$$

右腿每手保证金即为右腿实际的保证金，具体计算方法请参考股票期权保证金。

5.3.6. 保证金试算

保证金以及组合保证金优惠的计算比较复杂，为了方便使用ydExtendedApi的投资者以指定的价格预算保证金，易达提供了一些列方法供使用。

5.3.6.1. 报单保证金试算

投资者可以随时通过下列方法获取每手报单的期货保证金，试算过程没有计算大边保证金，仅仅是该报单自身的保证金。本方法不适用组合合约。

```
virtual double getMarginPerLot(const YDInstrument *pInstrument,int
hedgeFlag,int anyDirection,double openPrice,const YDAccount *pAccount=NULL)
```

上述方法中的参数如下表：

参数	说明
pInstrument	合约指针
hedgeFlag	投保标志。 YD_HF_Speculation=1：投机 YD_HF_Arbitrage=2：套利，只有中金所支持。 YD_HF_Hedge=3：套保
anyDirection	买卖方向，可以使用YD_D_Buy或YD_PD_Long表示买，YD_D_Sell或YD_PD_Short表示卖
openPrice	根据YDSystemParam中（Name, Target）为（MarginBasePrice, Options）的参数值决定使用的价格： YD_CBT_PreSettlementPrice：始终使用昨结算价，openPrice无效 YD_CBT_OpenPrice：使用openPrice指定的值 YD_CBT_LastPrice：始终使用最新价，openPrice无效 YD_CBT_MarketAveragePrice：始终使用市场平均成交价，openPrice无效 YD_CBT_MaxLastPreSettlementPrice：始终使用最新价和昨结算价的大者，openPrice无效
pAccount	设置为NULL

投资者可以随时通过下列方法获取每手报单的期权保证金，试算使用的价格是由API根据价格设置自行选择的。

```
virtual double getOptionsShortMarginPerLot(const YDInstrument
*pInstrument,int hedgeFlag,bool includePremium,const YDAccount
*pAccount=NULL)
```

上述方法中的参数如下表：

参数	说明
pInstrument	合约指针
hedgeFlag	投保标志，期货交易所与股票期权交易所的定义不同。 期货交易所如下： YD_HF_Speculation=1：投机 YD_HF_Arbitrage=2：套利，只有中金所支持。 YD_HF_Hedge=3：套保 股票期权交易所如下： YD_HF_Normal=1：普通 YD_HF_Covered=3：备兑
includePremium	是否需要包含权利金

参数	说明
pAccount	设置为NULL

5.3.6.2. 持仓保证金试算

以下方法可以试算在不同的价格条件下持仓的每手保证金，如果需要获取持仓完整的保证金需要自行乘以持仓量，对于期货持仓不计算大边保证金。与报单保证金试算不同，此处的openPrice并不受到YDSystemParam的影响，在计算时标的价格直接使用的是参数中openPrice指定的价格。

```
virtual double getMarginPerLot(const YDExtendedPosition *pPosition,double openPrice)
```

5.3.6.3. 组合保证金试算

以下方法可以试算尚不存在的组合，试算使用的价格是昨结算价，所以可能会与组合后实际收取的保证金有所区别。

```
virtual double getCombPositionMarginPerLot(const YDCombPositionDef *pCombPositionDef,const YDAccount *pAccount=NULL)
```

以下方法可以获取持仓明细真实的两腿保证金、节约保证金和组合保证金，
$组合保证金 = 左腿保证金 + 右腿保证金 - 节约保证金$，其中左右腿保证金分别是legMargins[0]和legMargins[1]，节约保证金是本方法的返回值。

```
virtual double getCombPositionMarginSaved(const YDExtendedCombPositionDetail *pCombPositionDetail,double legMargins[])
```

5.4. 期货持仓盈亏

$$期货多头持仓盈亏 = ((最新价 - 开仓价) \times 今持仓量 + (最新价 - 昨结算价) \times 昨持仓量) \times 合约乘数$$

$$期货空头持仓盈亏 = ((开仓价 - 最新价) \times 今持仓量 + (昨结算价 - 最新价) \times 昨持仓量) \times 合约乘数$$

持仓盈亏的刷新逻辑请参见资金刷新机制。

5.5. 期货平仓盈亏

$$期货多头平今仓盈亏 = (平仓价–开仓价) \times 合约乘数 \times 平仓手数$$

$$期货多头平昨仓盈亏 = (平仓价–昨结算) \times 合约乘数 \times 平仓手数$$

$$期货空头平今仓盈亏 = (开仓价–平仓价) \times 合约乘数 \times 平仓手数$$

$$期货空头平昨仓盈亏 = (昨结算–平仓价) \times 合约乘数 \times 平仓手数$$

平仓配对持仓明细的逻辑请参见持仓结构。

6. 交易

6.1. 支持的业务

6.1.1. 报单

报单指令是最常用的指令，投资者可以调用报单指令发起期货和期权的交易。可以用于报单的指令如下所示。

```
virtual bool insertOrder(YDInputOrder *pInputOrder,const YDInstrument *pInstrument,const YDAccount *pAccount=NULL)=0;
virtual bool insertMultiOrders(unsigned count,YDInputOrder inputOrders[],const YDInstrument *instruments[],const YDAccount *pAccount=NULL)=0;

// only available in ydExtendedApi
virtual bool checkAndInsertOrder(YDInputOrder *pInputOrder,const YDInstrument *pInstrument,const YDAccount *pAccount=NULL)=0;
virtual bool checkOrder(YDInputOrder *pInputOrder,const YDInstrument *pInstrument,const YDAccount *pAccount=NULL)=0;
```

6.1.1.1. 普通报单

首先介绍最普通的报单指令insertOrder，如果该函数返回false，则说明到柜台的网络发生了中断。

```
virtual bool insertOrder(YDInputOrder *pInputOrder,const YDInstrument *pInstrument,const YDAccount *pAccount=NULL)=0;
```

上述方法的参数说明如下：

参数	字段	说明
YDInputOrder	YDOrderFlag	填写YD_YOF_Normal
	OrderRef	客户报单编号，柜台在回报中会带回客户报单编号，投资者可以以此将报单和回报对应起来。
	Direction	YD_D_Buy：买 YD_D_Sell：卖

参数	字段	说明
	OffsetFlag	对于**普通合约**: YD_OF_Open：开仓 YD_OF_Close：平仓，适用于非上期所和能源所。若在上期所和能源所使用会被视作YD_OF_CloseYesterday YD_OF_CloseYesterday：平昨仓，适用于上期所和能源所。若在非上期所和能源所使用会被视作YD_OF_Close YD_OF_CloseToday：平今仓，适用于上期所和能源所。若在非上期所和能源所使用会被视作YD_OF_Close 对于**组合合约**: YD_OF_Open：同时开左右腿 YD_OF_Close：同时平左右腿 YD_OF_Open1Close2：开左腿平右腿 YD_OF_Close1Open2：平左腿开右腿
	HedgeFlag	YD_HF_Speculation：投机 YD_HF_Arbitrage：套利，只支持中金所 YD_HF_Hedge：套保
	OrderType	YD_ODT_Limit：限价单 YD_ODT_Market：市价单 YD_ODT_FAK：FAK单 YD_ODT_FOK：FOK单
	Price	填写价格，价格不能超过涨跌停板限制，易达不控制价格波动带
	OrderVolume	开平仓数量，受到以下限制: 不能超过合约的MaxMarketOrderVolume、MaxLimitOrderVolume 不能低于合约的MinMarketOrderVolume、MinLimitOrderVolume 不能超过持仓量限制，注意上交所和深交所需去除组合仓以后的持仓量
	OrderTriggerType	触发类型 YD_OTT_NoTrigger：无触发条件 YD_OTT_TakeProfit：止盈触发 YD_OTT_StopLoss：止损触发 目前支持大商所和广期所的触发单
	TriggerPrice	触发价格
YDInstrument		指定交易合约指针
YDAccount		填写NULL，表示使用当前API登陆账户

其中，易达系统中定义的指令类型OrderType是交易所指令类型条件的组合，针对每个交易所选择最贴近易达指令类型的组合进行设置，以下为各个交易所中的对照关系：

交易所	YD_ODT_Limit	YD_ODT_FAK	YD_ODT_Market	YD_ODT_FOK
CFFEX	FFEX_FTDC_TC_GFD FFEX_FTDC_OPT_LimitPrice FFEX_FTDC_VC_AV	FFEX_FTDC_TC_IOC FFEX_FTDC_OPT_LimitPrice FFEX_FTDC_VC_AV	FFEX_FTDC_TC_IOC FFEX_FTDC_OPT_AnyPrice FFEX_FTDC_VC_AV	FFEX_FTDC_TC_IOC FFEX_FTDC_OPT_LimitPrice FFEX_FTDC_VC_CV
SHFE	SHFE_FTDC_TC_GFD SHFE_FTDC_OPT_LimitPrice SHFE_FTDC_VC_AV	SHFE_FTDC_TC_IOC SHFE_FTDC_OPT_LimitPrice SHFE_FTDC_VC_AV	不支持，若使用则转换为： SHFE_FTDC_TC_IOC SHFE_FTDC_OPT_AnyPrice SHFE_FTDC_VC_AV	SHFE_FTDC_TC_IOC SHFE_FTDC_OPT_LimitPrice SHFE_FTDC_VC_CV
DCE	OT_LO OA_NONE	OT_LO OA_FAK	OT_MO OA_FAK	OT_LO OA_FOK

CZCE	FID_OrderType=0 //限价
	FID_MatchCondition=3 //当日有效
		FID_OrderType=0 //限价
		FID_MatchCondition=2 //即时部分成交
			FID_OrderType=1 //市价
			FID_MatchCondition=2 //即时部分成交
				FID_OrderType=0 //限价
				FID_MatchCondition=1 //即时全部成交

GFEX	OT_LO OA_NONE	OT_LO OA_FAK	OT_MO OA_FAK	OT_LO OA_FOK
SSE	OrderType=Limit TimeInForce=GFD	不支持，若使用则转换为 YD_ODT_FOK	OrderType=Market TimeInForce=IOC	OrderType=Limit TimeInForce=FOK
SZSE	OrderType=Limit TimeInForce=GFD MinQty=0	不支持，若使用则转换为 YD_ODT_FOK	OrderType=Market TimeInForce=IOC MinQty=0	OrderType=Limit TimeInForce=GFD MinQty=OrderVolume

6.1.1.2. 批量报单

批量报单可以一次性报不超过16个报单，可以满足客户多腿同时报单的需求。与多次调用普通报单相比，批量报单的整体性能并不占优，我们并不建议出于性能目的使用批量报单。下面我们从发送和接收两方面进行分析：

- 客户端发送时，批量报单的优点是可以减少API调用次数，但是需要把所有报单全部准备好以后才能进行发送，而普通报单却可以准备好一张立即发送一张，对于资源的利用率更高；而且批量报单在一个大包里面发出，发送时间更长
- 柜台接收时，批量报单也是顺序接收和处理的，这与多个普通报单的处理性能是基本相同的

```
virtual bool insertMultiOrders(unsigned count,YDInputOrder
inputOrders[],const YDInstrument *instruments[],const YDAccount
*pAccount=NULL)
```

批量报单全部发送成功返回true，部分或全部发送失败返回false。参数填写方法如下：

参数	说明
unsigned count	报单张数，最多不能超过16张
YDInputOrder inputOrders[]	报单数组，与合约指针数组一一对应
YDInstrument *instruments[]	交易合约指针数组
YDAccount	填写NULL，表示使用当前API登陆账户

易达柜台收到批量报单后，对每张报单的处理方式与普通报单的单张报单的处理方式是完全相同的：即每张报单都是独立事务，已经成功处理的单子不会因为后续单子风控失败而整体回退，同时排在前面的错单亦不会影响后面报单的处理。因此，批量报单的回报、撤单都与普通报单的相同，只不过会有多次回报回调。

6.1.1.3. 本地风控报单

在ydExtendedApi中，增加了一系列在客户端风控并报单的指令，这些指令在相对应的报单指令上增加了本地钱仓和风控检查，如果没有通过风控，则会直接返回false，错误原因查看YDInputOrder中的ErrorNo获取。相比原始报单指令，当指令较容易被柜台风控拦截时，本地风控报单指令可以更快的知道报单是否可以**大概率**通过柜台的钱仓和风控检查，而节约了与柜台往返的交互时间，相对应的，也会增加客户端报单的时间开销。因此，请根据实际情况选择使用相应的报单指令。

```
virtual bool checkAndInsertOrder(YDInputOrder *pInputOrder,const
YDInstrument *pInstrument,const YDAccount *pAccount=NULL)
```

使用本地风控报单时，YDInputOrder中的OrderRef是由API自行从1开始编码，即便用户给OrderRef赋值，也会被API覆盖掉。该OrderRef编码机制利用了ydExtendedApi中提供的连接号编码机制，即本地风控报单天然支持多连接号。具体内容请参考多连接。在调用函数返回后，可以在传入的YDInputOrder.OrderRef中找到本次报单使用的OrderRef，如果检查失败的，同样可以在YDInputOrder.ErrorNo中找到报单风控失败的原因。

通过本地风控检查的报单不代表一定能通过柜台的风控检查，可能但不限于以下原因：

- 行情波动较大，柜台和客户端刷新持仓盈亏和保证金的时机不一致，导致客户端资金可能足够但柜台不足的情况
- 投资者使用多个策略在同一个账户内交易，可能存在因为其他策略报单的影响而导致柜台检查失败的情况
- 部分风控措施只在柜台端检查

使用本地风控的批量报单指令时，只有全部通过风控检查才能报送，这与批量报单在柜台端的处理行为是不同的。

投资者也可以使用以下指令只检查而不报单：

```
virtual bool checkOrder(YDInputOrder *pInputOrder,const YDInstrument
*pInstrument,const YDAccount *pAccount=NULL)
```

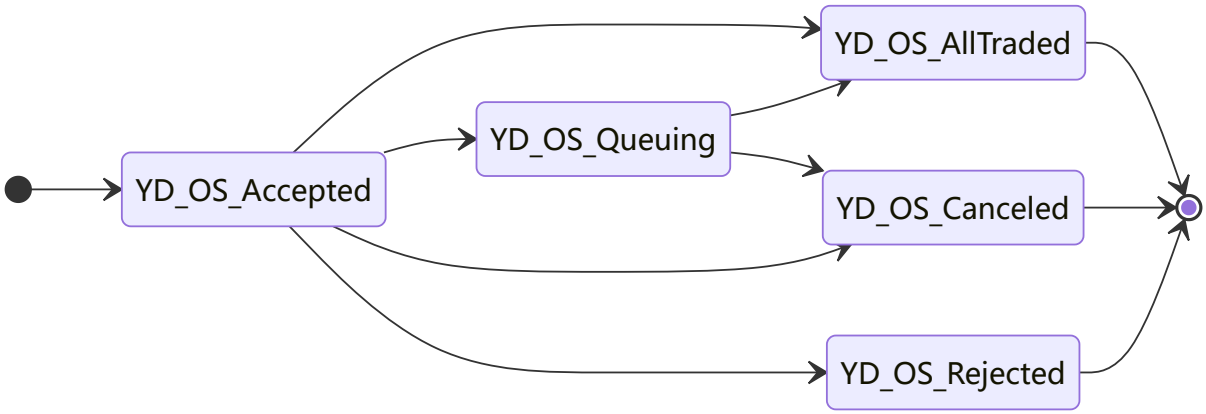
6.1.1.4. 报单回报

所有正确或者错误的交易所回报都将通过notifyOrder返回。

```
virtual void notifyOrder(const YDOrder *pOrder,const YDInstrument
*pInstrument,const YDAccount *pAccount)
```

若报单通过柜台风控检查并向交易所发送后，在收到交易所回报前，默认情况下是不会向投资者发送回报的。但是投资者可以请期货公司打开柜台回报功能以收取柜台回报，可以通过查看YDAccount中AccountFlag是否设置了YD_AF_NotifyOrderAccept来判断柜台回报是否开通成功。柜台回报可以作为柜台收到投资者报单申请的凭据，担心UDP报单丢失的投资者可使用本机制更早发现UDP丢包。柜台回报通过notifyOrder返回，其OrderStatus为YD_OS_Accepted，除了OrderSysID和InsertTime未赋值外，其他字段均已正确赋值，可以正常获取使用。除询价和大商所、广期所的期权对冲指令外，其他使用insertOrder系列报单指令报单的都会收到柜台回报。

每当收到交易所回报，易达柜台会据此的向投资者通过notifyOrder发送报单回报。回报的状态、数量和顺序与交易所具体行为有关，例如某些交易所FAK单、FOK单不会返回YD_OS_Queuing状态而就只有一个终态报单回报，再例如中金所的限价单如果在首次进入撮合队列时就全部成交的话那只会有一个全部成交的回报而不会有队列中的回报。交易所没有也不会承诺报单回报的行为，且有权随时改变该行为。但无论是易达还是交易所都将确保回报状态永远是按照报单状态机流转的，有鉴于此，投资者在开发策略程序时请不要依赖于交易所的回报行为，而是要做到无论以收到何种报单回报顺序都能正确处理。报单的状态机图如下所示。



notifyOrder的返回的参数说明如下，从YDInputOrder中带回的字段不再赘述：

参数	字段	说明
YDOrder	YDOrderFlag	固定为YD_YOF_Normal
	TradeVolume	报单的累计成交量
	OrderLocalID	柜台本地编号，本地编号从1开始按顺序全局统一编号，与席位无关。 若报单从其他柜台报入，如主席或者其他次席，则始终为-1
	OrderSysID	对于期货交易所，为交易所返回的系统报单号 对于现货交易所，为经过转换的虚拟交易所报单编号，要获取真实的交易所报单编号请参见下文 若交易所报单编号超过OrderSysID的最大长度，则会截取部分

参数	字段	说明
	LongOrderSysID	对于期货交易所，为交易所返回的系统报单号 对于现货交易所，为经过转换的虚拟交易所报单编号，要获取真实的交易所报单编号请参见下文 不会截取交易所报单编号，保留全精度
	InsertTime	从该交易日开始（17点整）到报单时间的秒数。请参考以下例子： 夜盘21点整：3600*(21-17)=14400 日盘9点整：3600*(24+9-17)=57600 周一日盘9点整：3600*(24+9-17)=5760 易达整数时间和标准时间的转换程序参见ydUtil.h中的string2TimeID和timeID2String
	InsertTimeStamp	从该交易日开始（17点整）到报单时间的毫秒数。请参考以下例子： 夜盘21点过500毫秒：3600*(21-17)*1000+500=14400500 日盘9点500毫秒：3600*(24+9-17)*1000+500=57600500 周一日盘9点500毫秒：3600*(24+9-17)*1000+500=5760500 易达时间戳时间和标准时间的转换程序参见ydUtil.h中的string2TimeStamp和timeStamp2String
	OrderTriggerStatus	报单触发状态 YD_OTS_NotTriggered：未触发 YD_OTS_Triggered：已触发
	OrderStatus	可能的状态如下，回报状态机见下图： YD_OS_Accepted：柜台接受，只会出现在柜台回报中 YD_OS_Queuing：队列中 YD_OS_AllTraded：全部成交 YD_OS_Canceled：撤单，包含部成部撤 YD_OS_Rejected：交易所错单
	ErrorNo	成功时为0，错误时为交易所错误号
YDInstrument		报单合约指针
YDAccount		当前API登陆账户

由于上交所、深交所使用的交易所报单编号、组合编号、行权编号并不是整数，易达无法直接把交易所返回的这些编号直接放到OrderSysID字段中，而是会把交易所原生编号经过数字化转换后填入OrderSysID，若想查询交易所原生编号，可以使用以下两种方法。

对使用ydApi和ydExtendedApi的投资者，可以通过YDListener的notifyIDFromExchange回调函数收到原生报单编号，只有当发生交易所编号发生转换才会收到此类回调，在期货交易所交易时不会收到此回调。

```
virtual void notifyIDFromExchange(const YDIDFromExchange
*pIDFromExchange,const YDExchange *pExchange)
```

返回的参数说明如下：

参数	字段	说明
YDIDFromExchange	IDType	ID类型，可能的值如下： YD_IDT_NormalOrderSysID：普通报单编号 YD_IDT_QuoteDerivedOrderSysID：报价衍生报单编号 YD_IDT_OptionExecuteOrderSysID：期权行权报单编号 YD_IDT_OptionAbandonExecuteOrderSysID：期权放弃行权报单编号 YD_IDT_RequestForQuoteOrderSysID：询价报单编号 YD_IDT_CombPositionOrderSysID：组合报单编号 YD_IDT_OptionExecuteTogether：合并行权报单编号 YD_IDT_MARK：组合报单编号 YD_IDT_Cover：备兑报单编号 YD_IDT_TradeID：成交编号 YD_IDT_CombPositionDetailID：组合明细编号 YD_IDT_QuoteSysID：报价编号
	IDInSystem	经过易达柜台转换的ID值，可能会被截断
	LongIDInSystem	经过易达柜台转换的全精度ID值
	IDFromExchange	交易所原生的ID值，最大长度24字节
YDExchange		交易所指针

对使用ydExtendedApi的投资者，还可以使用getIDFromExchange直接查询原生编号，查询时需要输入的参数的含义可以参见上表，返回的原生ID最大长度为24字节。

```
virtual const char *getIDFromExchange(const YDExchange *pExchange,int
idType,int idInSystem)
```

6.1.1.5. 扩展报单回报

若投资者使用了YDExtendedListener，那么可以通过notifyExtendedOrder在以下情况下会收到回报：

- notifyOrder回调时，无论是正确报单还是错误报单，无论柜台拒单还是交易所拒单
- 当YDExtendedOrder上的属性有任何变化时
- 当使用checkAndInsertOrder报单且发送成功后

```
virtual void notifyExtendedOrder(const YDExtendedOrder *pOrder)
```

由于YDExtendedOrder是继承自YDOrder，下面仅列举YDExtendedOrder的专属字段：

字段	说明
m_pInstrument	报单合约指针
m_pCombPositionDef	报单类型为YD_YOF_CombPosition组合或者解组时有效，为组合持仓定义指针
m_pAccount	报单所属的投资者指针
m_pInstrument2	报单类型为YD_YOF_OptionExecuteTogether合并行权时有效，为合并行权的第二个合约的指针

6.1.1.6. 成交回报

每当有成交产生就会通过notifyTrade返回成交回报，成交回报中的OrderRef、OrderLocalID和OrderSysID均可用于关联到具体报单。

```
virtual void notifyTrade(const YDTrade *pTrade,const YDInstrument
*pInstrument,const YDAccount *pAccount) {}
```

成交回报的返回值如下所示，与报单相同的字段不再重复：

参数	字段	说明
YDTrade	TradeID	交易所返回的成交编号 若交易所报单编号超过TradeID的最大长度，则会截取部分
	OrderLocalID	柜台本地编号，本地编号从1开始按顺序全局统一编号，与席位无关。 若报单从其他柜台报入，如主席或者其他次席，则始终为-1
	LongTradeID	全精度的交易所返回的成交编号
	OrderSysID	对于期货交易所，为交易所返回的系统报单号 对于现货交易所，为经过转换的虚拟交易所报单编号，要获取真实的交易所报单编号请参见下文 若交易所报单编号超过OrderSysID的最大长度，则会截取部分
	LongOrderSysID	对于期货交易所，为交易所返回的系统报单号 对于现货交易所，为经过转换的虚拟交易所报单编号，要获取真实的交易所报单编号请参见下文 不会截取交易所报单编号，保留全精度

参数	字段	说明
	OrderRef	报单编号
	OrderGroupID	报单所属报单组编号
	Price	成交价格
	Volume	成交数量
	Commission	成交手续费
	TradeTime	从该交易日开始（17点整）到成交时间的秒数。请参考以下例子： 夜盘21点整：3600*(21-17)=14400 日盘9点整：3600*(24+9-17)=57600 周一日盘9点整：3600*(24+9-17)=5760 TimeID和标准时间的转换程序参见ydUtil.h中的string2TimeID和timeID2String
	TradeTimeStamp	从该交易日开始（17点整）到成交时间的毫秒数。请参考以下例子： 夜盘21点过500毫秒：3600*(21-17)*1000+500=14400500 日盘9点500毫秒：3600*(24+9-17)*1000+500=57600500 周一日盘9点500毫秒：3600*(24+9-17)*1000+500=5760500 易达时间戳时间和标准时间的转换程序参见ydUtil.h中的string2TimeStamp和timeStamp2String
YDInstrument		成交合约指针
YDAccount		当前API登陆账户

6.1.1.7. 扩展成交回报

若投资者使用了YDExtendedListener，那么在任何notifyTrade回调的时候都会同时回调notifyExtendedTrade：

```
virtual void notifyExtendedTrade(const YDExtendedTrade *pTrade)
```

由于YDExtendedTrade是继承自YDTrade，下面仅列举YDExtendedTrade的专属字段：

字段	说明
m_pInstrument	成交合约指针
m_pAccount	成交所属的投资者指针

6.1.2. 撤单

投资者可以调用下列函数撤回处于队列中状态（OrderStatus为YD_OS_Queuing）的报单。

```
virtual bool cancelOrder(YDCancelOrder *pCancelOrder,const YDExchange *pExchange,const YDAccount *pAccount=NULL)
virtual bool cancelMultiOrders(unsigned count,YDCancelOrder cancelOrders[],const YDExchange *exchanges[],const YDAccount *pAccount=NULL)
```

6.1.2.1. 普通撤单

可以使用下列方法撤回单笔报单，如果该函数返回false，则说明到柜台的网络发生了中断。

```
virtual bool cancelOrder(YDCancelOrder *pCancelOrder,const YDExchange *pExchange,const YDAccount *pAccount=NULL)
```

上述方法的参数说明如下：

参数	字段	说明
YDCancelOrder	YDOrderFlag	填写待撤单的报单的报单类型
	OrderSysID	交易所报单编号，易达不支持OrderRef或者OrderLocalID撤单
	LongOrderSysID	全精度的交易所报单编号
	OrderGroupID	指定报单和报价所属的逻辑组，与OrderRef配合使用可实现OrderRef撤单
	OrderRef	与OrderGroupID配合使用可实现OrderRef撤单
YDExchange		撤单所在交易所
YDAccount		填写NULL，表示使用当前API登陆账户

柜台支持三种撤单方式，包括LongOrderSysID、OrderSysID和OrderRef。OrderRef撤单方式允许投资者收到回报前撤单，目前支持中金所、上期所、能源所、大商所、上交所、深交所。撤单方式的逻辑如下：

- 若OrderGroupID为0，则
  - 若LongOrderSysID非0，则使用LongOrderSysID撤单
  - 若LongOrderSysID为0，则使用OrderSysID撤单
- 若OrderGroupID非0，则使用OrderRef撤单

6.1.2.2. 批量撤单

批量撤单可以一次性撤不超过16个报单，其特性与批量报单类似。

```
virtual bool cancelMultiOrders(unsigned count,YDCancelOrder
cancelOrders[],const YDExchange *exchanges[],const YDAccount
*pAccount=NULL)
```

上述方法的参数说明如下：

参数	说明
unsigned count	撤单张数，最多不能超过16张
YDCancelOrder cancelOrders[]	撤单信息数组，与交易所指针数组一一对应
YDExchange *exchanges[]	对应于每个撤单信息所在的交易所指针数组
YDAccount	填写NULL，表示使用当前API登陆账户

6.1.2.3. 撤单回报

如果撤单失败，无论是柜台拒绝撤单还是交易所拒绝撤单，都会通过如下回调返回，请检查YDFailedCancelOrder中的ErrorNo获取撤单失败的原因，通常是该报单已经完成成交或者已撤单的错误：

```
virtual void notifyFailedCancelOrder(const YDFailedCancelOrder
*pFailedCancelOrder,const YDExchange *pExchange,const YDAccount *pAccount)
{}
```

如果撤单成功，那么被撤报单会通过notifyOrder返回其最新状态，具体请参阅报单回报。

6.1.3. 备兑

6.1.3.1. 现货锁定与解锁

上交所股票期权备兑开仓和普通转备兑前，需要首先锁定现货仓位。深交所执行备兑业务时其交易系统内部执行了冻结，因此可以直接开展备兑业务，无需事先冻结。

请按照下列参数说明，使用insertOrder指令向交易所发起锁定和解锁操作。

参数	字段	说明
YDInputOrder	YDOrderFlag	填写YD_YOF_FreezeUnderlying，说明该报单是现货锁定业务
	OrderRef	客户报单编号，详情请参考普通报单中关于OrderRef的说明
	Direction	YD_D_Freeze：锁定 YD_D_Unfreeze：解锁

参数	字段	说明
	OrderVolume	填写锁定和解锁的数量
	HedgeFlag	填写YD_HF_Covered
YDInstrument		指定待锁定或解锁的现货合约指针
YDAccount		填写NULL，表示使用当前API登陆账户

6.1.3.2. 备兑开平仓

请按照下列参数说明，使用insertOrder指令向交易所发起备兑开仓和平仓操作。

参数	字段	说明
YDInputOrder	YDOrderFlag	填写YD_YOF_Normal
	OrderRef	客户报单编号，详情请参考普通报单中关于OrderRef的说明
	Direction	YD_D_Buy：买 YD_D_Sell：卖
	OffsetFlag	YD_OF_Open：开仓 YD_OF_Close：平仓
	HedgeFlag	填写YD_HF_Covered
	OrderType	可以根据需要选择填写以下值： YD_ODT_Limit：限价单 YD_ODT_Market：市价单 YD_ODT_FOK：FOK单
	Price	填写价格
	OrderVolume	开平仓数量
YDInstrument		指定备兑开平仓的期权合约指针
YDAccount		填写NULL，表示使用当前API登陆账户

6.1.3.3. 普通与备兑转换

请按照下列参数说明，使用insertOrder指令向交易所发起普通转备兑或者备兑转普通的操作。请注意，上交所和深交所均支持普通转备兑业务，但是只有深交所支持备兑转普通业务。

参数	字段	说明
YDInputOrder	YDOrderFlag	填写YD_YOF_Cover
	OrderRef	客户报单编号，详情请参考普通报单中关于OrderRef的说明

参数	字段	说明
	Direction	YD_D_Normal2Covered：普通转备兑 YD_D_Covered2Normal：备兑转普通，仅适用于深交所
	HedgeFlag	填写YD_HF_Covered
	OrderVolume	转换数量
YDInstrument		指定待转换的期权合约指针
YDAccount		填写NULL，表示使用当前API登陆账户

6.1.4. 询价

对于上期所、能源所、中金所、大商所、郑商所、广期所，可参照下列参数说明，使用insertOrder发送询价请求。询价必须针对可以询价的合约，否则没有效果。

> 上期所、能源所将询价纳入信息量计算，为防止投资者信息量意外超出后被收取高额的手续费，易达提供了信息量风控。当信息量触发信息量风控设置的阈值上限后会禁止投资者交易该产品，包括询价。

参数	字段	说明
YDInputOrder	YDOrderFlag	填写YD_YOF_RequestForQuote
	Direction	填写YD_D_Buy
	OffsetFlag	填写YD_OF_Open
	HedgeFlag	填写YD_HF_Speculation
	OrderType	填写YD_ODT_Limit
YDInstrument		指定待询价的合约指针
YDAccount		填写NULL，表示使用当前API登陆账户

即便YDAccount中的AccountFlag设置了柜台回报YD_AF_NotifyOrderAccept，询价指令也不会发送柜台回报。

为减少不必要的流量，询价回报notifyRequestForQuote只在做市商系统（柜台授权文件中含有MarketMaker标志）中通知，普通系统无法收到询价回报。

```
virtual void notifyRequestForQuote(const YDRequestForQuote
*pRequestForQuote,const YDInstrument *pInstrument)
```

以下为notifyRequestForQuote中返回信息的说明。

参数	字段	说明
YDRequestForQuote	RequestTime	从该交易日开始（17点整）到询价时间的秒数。请参考以下例子： 夜盘21点整：3600*(21-17)=14400 日盘9点整：3600*(24+9-17)=57600 周一日盘9点整：3600*(24+9-17)=5760 易达整数时间和标准时间的转换程序参见ydUtil.h中的string2TimeID和timeID2String
	RequestForQuoteID	询价编号。若返回长度过长，则会被截断
	LongRequestForQuoteID	全精度的询价编号
YDInstrument		询价合约指针

在ydExtendedApi中，易达提供了主动查询询价号的方法，使用方法如下：

```
virtual const YDExtendedRequestForQuote *getRequestForQuote(const
YDInstrument *pInstrument)
```

上述方法的返回值YDExtendedRequestForQuote是YDRequestForQuote的子类，因此可以直接读取询价时间RequestTime和RequestForQuoteID询价编号。若查询的查阅不存在询价，那么上述方法将返回NULL。

6.1.5. 报价

易达支持上期所、能源所、中金所、大商所、广期所、郑商所、上交所、深交所的做市商报价业务，报价指令只能在打开了做市商功能的易达柜台中使用（柜台授权文件中含有MarketMaker标志），可以通过查看SystemParam中的MarketMaker标志查看当前柜台是否支持报价功能，详情参见系统参数。

6.1.5.1. 普通报价

做市商可以使用下列方法发送单张报价单，如果该函数返回false，则说明到柜台的网络发生了中断。

```
virtual bool insertQuote(YDInputQuote *pInputQuote,const YDInstrument
*pInstrument,const YDAccount *pAccount=NULL)
```

上述方法的参数说明如下：

参数	字段	说明
YDInputQuote	OrderRef	客户报单编号，详情请参考普通报单中关于OrderRef的说明
	BidOffsetFlag	买开平标志

参数	字段	说明
	BidHedgeFlag	买投保标志 中金所：交易所接口实际只支持两侧投保标志相同，易达不进行检查，但是报入交易所时使用买入方的投保标志对应的客户号 其他交易所：买卖方投保标志可以不同
	AskOffsetFlag	卖开平标志
	AskHedgeFlag	卖投保标志
	BidPrice	买价格，买价必须低于卖价
	AskPrice	卖价格
	BidVolume	买数量。根据YDExchange.QuoteVolumeRestriction有不同的买卖数量限制： 0（大商所和广期所）：BidVolume和AskVolume至少一个大于0，另一个大于等于0 1（中金所、上交所、深交所）：BidVolume和AskVolume必须大于0，但是可以不等 2（上期所、能源所、郑商所）：BidVolume和AskVolume必须相等且大于0
	AskVolume	卖数量
	OrderRef	客户报单编号
	YDQuoteFlag	报价标志 YD_YQF_ResponseOfRFQ：自动填写询价号表示应价，支持的有上期所、能源所、大商所、广期所、郑商所，其他交易所无询价号
YDInstrument		指定待报价的合约指针
YDAccount		填写NULL，表示使用当前API登陆账户

在发起报价请求后，为了能统一所有交易所的报价业务，易达柜台会为该报价生成对应的报单，这些报单的YDOrderFlag为YD_YOF_QuoteDerived，如果是双边报价会产生两笔衍生报单，如果是单边报价则只会产生一笔衍生报单。衍生的报单只存在于易达内部，并不会发送到交易所，报价单的成交、报单状态将会反映在衍生报单上面，而报价单自身是没有状态的。衍生报单YDOrder的字段与报价YDQuote字段的关系如下所示：

- 买卖衍生报单的Direction分别对应YD_D_Buy和YD_D_Sell
- 买卖衍生报单的OffsetFlag分别对应BidOffsetFlag和AskOffsetFlag
- 买卖衍生报单的HedgeFlag分别对应BidHedgeFlag和AskHedgeFlag
- 买卖衍生报单的Price分别对应BidPrice和AskPrice

- 买卖衍生报单的OrderVolume分别对应BidVolume和AskVolume
- 买卖衍生报单的OrderRef都为报价单的OrderRef
- 买卖衍生报单的YDOrderFlag都为YD_YOF_QuoteDerived
- 买衍生报单的OrderLocalID按照普通报单顺序编号，卖衍生报单OrderLocalID一定是买衍生报单OrderLocalID+1
- 买卖衍生报单的OrderSysID分别对应BidOrderSysID和AskOrderSysID

6.1.5.2. 批量报价

批量报价可以一次性报不超过12个报价，其特性与批量报单类似。

```
virtual bool insertMultiQuotes(unsigned count,YDInputQuote
inputQuotes[],const YDInstrument *instruments[],const YDAccount
*pAccount=NULL)
```

上述方法的参数说明如下：

参数	说明
unsigned count	报价单张数，最多不能超过12张
YDInputQuote inputQuotes[]	报单数组，与合约指针数组一一对应
const YDInstrument *instruments[]	对应于每个报价的交易所指针数组
YDAccount	填写NULL，表示使用当前API登陆账户

6.1.5.3. 报价回报

所有正确或者错误的交易所报价回报会通过notifyQuote返回。

若报价被广期所拒绝的，在以下两种情况和其他交易所的回报有所区别：

- 如果投资者使用易达的报价接口insertQuote报入单边报价，易达柜台会通过交易所的普通报单接口报入交易所，那么就会同时收到notifyQuote和notifyOrder回调通知错误，此时应检查YDQuote中的ErrorNo检查错误原因
- 如果投资者使用易达的报价接口insertQuote报入**非应价双边报价**，易达柜台会通过交易所的普通报单接口报入交易所，如果产生一边成功，一边失败的情况，那么就会收到notifyQuote报告报价错误，此时应检查YDQuote中的ErrorNo检查错误原因，同时会通过notifyOrder返回的一个成功的衍生报单回报和一个错误的衍生报单回报，此时请将报单成功的衍生报单作为普通报单处理，以免该报单得不到正确管理

若报价成功，则会通过以下回调函数收到通知，其中：

- notifyQuote：每个报价单只有一个回报，表示交易所已经收到了报价请求，不保证与notifyOrder之间的次序
- notifyOrder：每次衍生报单状态发生变化就会收到该回调，其状态机同普通报单

- notifyTrade：同普通报单的报价，每当报价有成交产生就会有成交回报，可以通过OrderLocalID关联到衍生报单，可以通过OrderSysID关联到报价单

```
virtual void notifyQuote(const YDQuote *pQuote,const YDInstrument *pInstrument,const YDAccount *pAccount) {}
virtual void notifyOrder(const YDOrder *pOrder,const YDInstrument *pInstrument,const YDAccount *pAccount) {}
virtual void notifyTrade(const YDTrade *pTrade,const YDInstrument *pInstrument,const YDAccount *pAccount) {}
```

其中，notifyOrder和notifyTrade同普通报单，notifyQuote中的YDQuote是YDInputQuote的子类，除了常规的RealConnectionID和ErrorNo，YDQuote新增信息如下：

参数	字段	说明
YDQuote	QuoteSysID	当ErrorNo为YD_ERROR_InvalidGroupOrderRef时，表示当前柜台已收到的最大OrderRef；否则表示报价编号，用于撤报价。若交易所返回值超长，则会被截断
	BidOrderSysID	买报价的OrderSysID，卖单边报价时为0。若交易所返回值超长，则会被截断
	AskOrderSysID	卖报价的OrderSysID，买单边报价时为0。若交易所返回值超长，则会被截断
	RequestForQuoteID	当YDQuoteFlag为YD_YQF_ResponseOfRFQ时，记录了报单响应的询价号，否则为0。若交易所返回值超长，则会被截断
	LongQuoteSysID	全精度的报价编号，用于撤报价
	LongBidOrderSysID	全精度的买报价的OrderSysID，卖单边报价时为0
	LongAskOrderSysID	全精度的卖报价的OrderSysID，买单边报价时为0
	LongRequestForQuoteID	全精度的当YDQuoteFlag为YD_YQF_ResponseOfRFQ时，记录了全精度的报单响应的询价号，否则为0
YDInstrument		报价合约指针
YDAccount		当前API登陆账户指针

6.1.5.4. 扩展报价回报

若投资者使用了YDExtendedListener，那么可以通过notifyExtendedQuote在以下情况下会收到回报：

- notifyQuote回调时，无论是正确报价还是错误报价，无论柜台拒单还是交易所拒单

- 当YDExtendedQuote上的属性有任何变化时
- 当使用checkAndInsertQuote报单且发送成功后

```
virtual void notifyExtendedQuote(const YDExtendedQuote *pQuote)
```

由于YDExtendedQuote是继承自YDQuote，下面仅列举YDExtendedQuote的专属字段：

字段	说明
BidOrderFinished	买腿对应的衍生报单是否已经完成 当衍生报单是撤单、全成、拒绝时被认为是完成 若报价买量为0，则直接被认为是完成
AskOrderFinished	卖腿对应的衍生报单是否已经完成 当衍生报单是撤单、全成、拒绝时被认为是完成 若报价卖量为0，则直接被认为是完成
m_pInstrument	报价合约指针
m_pAccount	报价所属的投资者指针

6.1.6. 撤报价

投资者可以调用下列方法撤回未完成的报价。

```
virtual bool cancelQuote(YDCancelQuote *pCancelQuote,const YDExchange
*pExchange,const YDAccount *pAccount=NULL)
virtual bool cancelMultiQuotes(unsigned count,YDCancelQuote
cancelQuotes[],const YDExchange *exchanges[],const YDAccount
*pAccount=NULL)
```

中金所、大商所、广期所、上交所、深交所支持撤回衍生报单的方式撤回双边报价的一边，上期所、能源所、郑商所不支持。撤回衍生报单的方式同普通报单，请参考撤单的相应内容。与撤回报价类似，撤回衍生报单时只会有报单回调notifyOrder，而没有报价回调notifyQuote。

6.1.6.1. 普通撤报价

可以使用下列方法撤回单笔报价，如果该函数返回false，则说明到柜台的网络发生了中断。

```
virtual bool cancelQuote(YDCancelQuote *pCancelQuote,const YDExchange
*pExchange,const YDAccount *pAccount=NULL)
```

上述方法的参数说明如下：

参数	字段	说明
YDCancelQuote	QuoteSysID	报价编号

参数	字段	说明
	LongQuoteSysID	全精度的交易所报价编号
	OrderGroupID	指定报价所属的逻辑组，与OrderRef配合使用可实现OrderRef撤报价
	OrderRef	与OrderGroupID配合使用可实现OrderRef撤报价
YDExchange		待撤回报价的交易所指针
YDAccount		填写NULL，表示使用当前API登陆账户

柜台支持三种撤报价方式，包括LongOrderSysID、OrderSysID和OrderRef。OrderRef撤单方式允许投资者收到回报前撤单，目前支持中金所、上期所、能源所、大商所、上交所、深交所。撤单方式的逻辑如下：

- 若OrderGroupID为0，则
  - 若LongQuoteSysID非0，则使用LongQuoteSysID撤报价
  - 若LongQuoteSysID为0，则使用OuoteSysID撤报价
- 若OrderGroupID非0，则使用OrderRef撤报价

6.1.6.2. 批量撤报价

批量撤报价可以一次性撤不超过16个报价，其特性与批量报单类似。

```
virtual bool cancelMultiQuotes(unsigned count,YDCancelQuote
cancelQuotes[],const YDExchange *exchanges[],const YDAccount
*pAccount=NULL)
```

上述方法的参数说明如下：

参数	说明
unsigned count	撤报价张数，最多不能超过16张
YDCancelQuote cancelQuotes[]	撤报价信息数组，与交易所指针数组一一对应
YDExchange *exchanges[]	对应于每个撤报价信息所在的交易所指针数组
YDAccount	填写NULL，表示使用当前API登陆账户

6.1.6.3. 撤报价回报

如果撤报价成功，则通过notifyOrder回调返回各个衍生报单的状态，请注意，撤回报价时**不会**有notifyQuote回调。

```
virtual void notifyOrder(const YDOrder *pOrder,const YDInstrument
*pInstrument,const YDAccount *pAccount) {}
```

如果撤报价失败，则通过notifyFailedCancelOrder回调返回错误信息，此时应检查YDQuote中的ErrorNo检查错误原因：

```
virtual void notifyFailedCancelQuote(const YDFailedCancelQuote
*pFailedCancelQuote,const YDExchange *pExchange,const YDAccount *pAccount)
```

6.1.7. 报价顶单

上期所、能源所、大商所、广期所、郑商所、上交所、深交所支持报价顶单业务（以报待撤），即可以通过新的报价顶替之前的相同合约的报价，可以减少撤单次数。

投资者使用顶单业务报价时，可以使用普通报价的方法直接提交新的报价，回调情况等同于一次撤报单和一次报价产生的回报，因此请参考报价和撤报价相关内容。

6.1.8. 组合与解组合

大商所、广期所、上交所、深交所没有大边保证金业务，只能通过组合持仓优惠保证金。郑商所既有同合约单向大边保证金，也可以通过组合持仓优惠保证金。

对于大商所和广期所，结算时会进行自动组合持仓，因此开盘前这些持仓均处于组合状态。如果盘中有新开仓符合组合持仓定义且需要组合的，可以使用组合指令予以组合。平仓时，大商所和广期所组合自动解组，无需事先解组。

对于郑商所，结算时备兑会自动组合保证金，其他类型需要投资者申请，因此开盘前备兑和申请成功的持仓会处于组合状态，盘中时按照对锁（同合约大边）自动优惠保证金。平仓时，郑商所组合自动解组，无需事先解组。

对于上交所和深交所，结算时不会自动组合，只能在盘中通过组合指令进行组合。平仓时，上交所和深交所的持仓必须先解组，然后平仓，处于组合的持仓无法被平仓。

易达提供了3种不同的方法发送组合指令，分别是：

- 原生指令：ydApi和ydExtendedApi的用户可以调用insertCombPositionOrder发送原生组合和解组指令，ydExtendedApi的用户可以调用checkAndInsertCombPositionOrder发送原生组合和解组指令
- 自动指令：ydExtendedApi的用户可以调用autoCreateCombPosition发起自动组合指令
- 自动工具：运行自动组合工具ydAutoCP定时发送组合指令，该工具可以从这里下载或者向经纪商索取。

6.1.8.1. 原生指令

ydApi和ydExtendedApi的用户可以调用insertCombPositionOrder发送原生组合和解组指令，ydExtendedApi的用户可以调用checkAndInsertCombPositionOrder发送原生组合和解组指令。与其他组合方法相比，原生指令提供了最大的自由度，投资者可以不受优先级约束随意选择需要组合的指令，而且原生指令也是唯一支持解组合的指令。同时，自动指令和自动工具都是基于原生指令开发的。

```
virtual bool insertCombPositionOrder(YDInputOrder *pInputOrder,const YDCombPositionDef *pCombPositionDef,const YDAccount *pAccount=NULL)
virtual bool checkAndInsertCombPositionOrder(YDInputOrder *pInputOrder,const YDCombPositionDef *pCombPositionDef,const YDAccount *pAccount=NULL)
```

checkAndInsertCombPositionOrder与insertCombPositionOrder的参数是相同的，区别在于checkAndInsertCombPositionOrder相对insertCombPositionOrder增加了API端对输入参数的合法性检查，如果合法性检查失败，则checkAndInsertCombPositionOrder会直接返回false，错误原因查看YDInputOrder中的ErrorNo获取，而不需要等待柜台检查后返回错误，这些合法性检查包括：

- 买卖方向、投保标志、组合数量的取值合法性检查
- 是否有足够的持仓进行组合，是否有足够的组合持仓进行解组

API本地合法性检查可以减少因字段不合法而被退回的时间，但是会在每个组合指令上增加额外的开销。好在组合指令并非是性能敏感型指令，因此我们推荐所有希望使用原生指令的ydExtendedApi用户都能使用checkAndInsertCombPositionOrder来发起组合指令。

insertCombPositionOrder和checkAndInsertCombPositionOrder的调用方法如下所示。

参数	字段	说明
YDInputOrder	YDOrderFlag	填写YD_YOF_CombPosition
	OrderRef	客户报单编号，详情请参考普通报单中关于OrderRef的说明
	Direction	YD_D_Make：组合 YD_D_Split：解组
	OrderType	填0
	HedgeFlag	填写YD_HF_Speculation
	OrderVolume	组合需要或者解组的数量。 组合时需要两腿的未组合持仓量都大于组合数量。 解组时需要组合持仓量大于解组数量。
	CombPositionDetailID	组合明细ID，只有当在上交所和深交所解组时需要填写，其他情况填0
YDCombPositionDef		待组合的组合持仓定义指针，请参考组合持仓定义获取。

参数	字段	说明
YDAccount		填写NULL，表示使用当前API登陆账户

组合成功或者失败后，会通过以下回调接口通知投资者。

```
virtual void notifyCombPositionOrder(const YDOrder *pOrder,const
YDCombPositionDef *pCombPositionDef,const YDAccount *pAccount) {}
```

6.1.8.1.1. 扩展组合回报

若投资者使用了YDExtendedListener，那么可以通过notifyExchangeCombPositionDetail在以下情况下会收到组合明细回报：

- 接收日初组合明细数据时，请参见日初组合持仓
- 当组合或者解锁指令成功导致组合明细发生变化时

```
virtual void notifyExchangeCombPositionDetail(const
YDExtendedCombPositionDetail *pCombPositionDetail)
```

YDExtendedCombPositionDetail的结构体请参见日初组合持仓。

6.1.8.2. 自动指令

对于ydExtendedApi的用户而言，大多数情况下可以调用autoCreateCombPosition来自动完成组合。目前，自动指令只支持大商所和广期所，而上交所和深交所咋平仓时必须要先解组，而解组必须使用原生指令，因此，对于上交所和深交所的组合和解组，请直接使用原生指令。

```
virtual const YDCombPositionDef *autoCreateCombPosition(const int
*combTypes)
```

每次调用autoCreateCombPosition，会首先按照combTypes指定的组合类型顺序，同一个组合类型中按照YDCombPositionDef.Priority（越小表示优先级越高）指定的优先级顺序，逐一检查可以组合的持仓，一旦检查到可以组合的持仓，就发送组合指令到柜台。请注意，每次调用autoCreateCombPosition最多只会组合一套，在生产实践中，可以用一个单独的线程按照一定的间隔持续调用本函数，如果找到需要组合的持仓并发送成功的，函数返回发送组合的组合持仓定义，如果没有找到需要组合的持仓，函数返回NULL。

autoCreateCombPosition的调用方法如下所示。

参数	说明
const int* combTypes	需要参与组合的类型数组，以-1结尾。组合时按照数组顺序逐个类型组合。 当为NULL时，默认为按照大商所所有组合持仓类型的顺序，即从YD_CPT_DCE_FuturesOffset到YD_CPT_DCE_SellOptionsCovered

组合成功或者失败后，会通过以下回调接口通知投资者。

```
virtual void notifyCombPositionOrder(const YDOrder *pOrder,const
YDCombPositionDef *pCombPositionDef,const YDAccount *pAccount) {}
```

6.1.8.3. 自动工具

早期易达提供了独立的自动组合工具ydAutoCP供投资者使用，该工具通过命令行启动，启动后会按照一定的时间间隔搜索可以组合的持仓并发送组合指令。鉴于ydAutoCP有诸多限制条件和使用上的不便，我们更推荐投资者使用API来发送组合指令。出于与自动指令类似相同的原因，ydAutoCP也只支持大商所的自动组合。

ydAutoCP的配置文件样ydACP.ini的例如下：

```
AppID=yd_dev_1.0
AuthCode=ecbf8f06469eba63956b79705d95a603

AccountID=001
Password=001

# time ranges for auto combine position , can be multiple, no such setting
indicates selecting at any time. Default is no setting
TimeRange=21:00:05-23:29:55
TimeRange=09:00:05-10:14:55
TimeRange=10:30:05-11:29:55
TimeRange=13:30:05-15:29:55

# refresh period, in seconds, default is 10, must be greater than 1
RefreshPeriod=10

# directory to hold log files, default is "ACPLog"
LogDir=ACPLog

# Comb positions types to be processed. If not given, default is all
types. Optional values are as follow:
# YD_CPT_DCE_FuturesOffset=0
# YD_CPT_DCE_OptionsOffset=1
# YD_CPT_DCE_FuturesCalendarSpread=2
# YD_CPT_DCE_FuturesProductSpread=3
# YD_CPT_DCE_BuyOptionsVerticalSpread=4
# YD_CPT_DCE_SellOptionsVerticalSpread=5
# YD_CPT_DCE_OptionsStraddle=7
# YD_CPT_DCE_OptionsStrangle=8
# YD_CPT_DCE_BuyOptionsCovered=9
# YD_CPT_DCE_SellOptionsCovered=10
CombPositionType=0,1,2,3,4,5,7,8,9,10
```

除了几个显而易见的参数外，核心参数的说明如下：

- TimeRange指定了系统运行的时间。因为ydAutoCP遇到组合错误时会禁止该组合持仓定义参与后续组合，如果在非交易时间发送组合单会导致在此期间的组合持仓定义全部被禁止，因此，在配置系统时必须避开交易所的非交易时间段。同时，运行ydAutoCP的操作系统必须正确对时。
- CombPositionType与autoCreateCombPosition的参数combTypes的含义是完全相同的，请参考自动指令中的相关内容。

ydAutoCP的启动方式为，其中config.txt为易达API的配置文件，而ydAutoCP的配置文件ydACP.ini是默认加载的，因此该文件名不可以被修改：

```
./ydAutoCP config.txt
```

一旦发现系统发生了组合持仓定义被禁止的情况，如果想要使得该定义重新生效，必须重启ydAutoCP。组合持仓定义被禁止可以通过查看日志检查，相关错误信息样例为：

```
error in handling (pg2302&-eg2303,1) for account 12345678
```

6.1.9. 期权对冲

对于上期所和能源所，可以指定待持仓对冲的数量，请参考以下方法使用insertOrder指令向交易所发起和撤销期权对冲请求。上期所和能源所为不同的投资者设定了不同的对冲默认值：对于普通投资者，其期权仓位默认是不对冲的，投资者可以申请对冲或者放弃申请对冲；对于做市商，其期权仓位默认是对冲的，做市商可以申请不对冲或者放弃申请不对冲。对于同一投资者、同一合约、同一OrderType，只保留最近申请的记录。

在当前易达系统实现中，上期所和能源所行权获得的期货仓位是自对冲的，这一行为是固化在向交易所发送的行权指令之中的，目前没有修改方式。

参数	字段	说明
YDInputOrder	YDOrderFlag	填写YD_YOF_OptionSelfClose
	OrderRef	客户报单编号，详情请参考普通报单中关于OrderRef的说明
	OrderType	YD_ODT_CloseSelfOptionPosition：请求对冲期权，供普通投资者使用 YD_ODT_ReserveOptionPosition：请求不对冲期权，供做市商使用 YD_ODT_SellCloseSelfFuturesPosition：请求待卖期权履约后对冲期货仓位
	HedgeFlag	投保标志 YD_HF_Speculation=1：投机 YD_HF_Hedge=3：套保
	OrderVolume	>0：发起并指定请求的数量 0：放弃对应（同一投资者、同一合约、同一OrderType）的请求

参数	字段	说明
YDInstrument		指定待对冲的期权合约指针
YDAccount		填写NULL，表示使用当前API登陆账户

对于大商所和广期所，不可以指定待持仓对冲的数量。在当前易达系统实现中，大商所和广期所行权获得的期货仓位是自对冲的，这一行为是固化在向交易所发送的行权指令之中的，目前没有修改方式。

请参考以下方法使用insertOrder指令向交易所发起和撤销期权对冲请求。

参数	字段	说明
YDInputOrder	YDOrderFlag	填写YD_YOF_Mark
	OrderRef	始终填写0 回报中的OrderRef始终为-1，和请求中的OrderRef无法对应
	Direction	YD_D_Buy：发起对冲请求 YD_D_Sell：撤销对冲请求
	OrderType	YD_ODT_PositionOffsetMark：持仓对冲 YD_ODT_CloseFuturesPositionMark：履约对冲，跨日有效
	HedgeFlag	投保标志 YD_HF_Speculation=1：投机 YD_HF_Hedge=3：套保
YDInstrument		持仓对冲时，指定的待对冲合约指针 履约对冲时，指定对应交易所的任意合约指针
YDAccount		填写NULL，表示使用当前API登陆账户

由于大商所和广期所对冲指令的特殊性，易达无法关联发往交易所的请求和回报，易达在收到交易所的回报后，会产生一个特殊的回报。因此，投资者在发送大商所和广期所的对冲指令后，不要在客户端保留报单或者等待与之匹配的回报，而是在收到YDOrderFlag为YD_YOF_Mark的报单回报后，根据合约或合约所在交易所、OrderType和OrderStatus来区分具体发生的业务和成功与否。也是基于上述同样的原因，即便YDAccount中的AccountFlag设置了柜台回报YD_AF_NotifyOrderAccept，大商所和广期所的对冲指令也不会发送柜台回报。

请注意，大商所和广期所的履约对冲是设置在账户层面的，即只要提交过履约对冲请求，针对该账户的所有期权在履约后都会对冲。出于简化API考虑，在发起履约对冲指令时，请任意指定一个属于大商所或者广期所的合约，便于易达柜台通过该合约找到要发起履约对冲指令的交易所。在回报中，交易所在返回时并不包含具体合约，因此，易达会将对应交易所的首个合约随回报传回，投资者只需要获取该合约的交易所即可。

注意，大商所和广期所的对冲指令无法选择席位报单，易达交易系统会一律覆盖指定为从首席位发出。这么处理的原因是这类报单既没有系统报单号，也没有本地报单号，所以无法实现任何识别重复接收的机制。如果使用全管理席位叠加全收模式流水，所有席位都会收到回报，会造成易达柜台产生无法关联的报单回报，所以，目前的实现是一律用首席位发送和接收。但是这也带来一个问题，如果首席位不是公用席位，那么对于不能使用首席位的投资者，就无法发出本指令，因此对于大商所和广期所，不能将首席位配置为专属席位。

下表详细描述了通过notifyOrder收到的回报的赋值情况：

参数	字段	说明
YDOrder	YDOrderFlag	YD_YOF_Mark
	Direction	YD_D_Buy：发起对冲请求 YD_D_Sell：撤销对冲请求
	OrderType	YD_ODT_PositionOffsetMark：持仓对冲 YD_ODT_CloseFuturesPositionMark：履约对冲
	HedgeFlag	投保标志 YD_HF_Speculation=1：投机 YD_HF_Hedge=3：套保
	OrderRef	-1
	OrderLocalID	-1
	OrderSysID	0
	InsertTime	-1
	OrderStatus	成功：YD_OS_AllTraded 失败：YD_OS_Rejected
	ErrorNo	成功时为0，错误时为交易所错误号
YDInstrument		持仓对冲时，返回指令中指定的合约指针 履约对冲时，返回对应交易所的首个合约
YDAccount		当前API登陆账户

中金所和郑商所没有期权对冲业务。

6.1.10. 期权行权

对于上期所、能源所、大商所、郑商所、广期所的美式期权以及上交所、深交所的期权，可以使用insertOrder向交易所发起行权申请，使用cancelOrder撤销行权申请。上期所、能源所、大商所、郑商所、广期所对美式期权的行权指令可以在任意交易日发出，沪深两市的行权指令必须与行权日当日发出。

为便于编写程序，易达为是否支持期权行权提供了参数化表示，使用YDExchange.OptionExecutionSupport来表示对期权行权不同程度的支持：

- 0：不支持。目前中金所属于这个分类。
- 1：支持但不包含风控，所谓不风控就是不做钱仓检查和冻结。目前没有交易所属于这个分类。
- 2：支持且包含风控，所谓风控就是做钱仓检查和冻结。目前上期所、能源所、大商所、郑商所、广期所、上交所、深交所属于这个分类。

下面介绍期权行权的参数填写方法。

参数	字段	说明
YDInputOrder	YDOrderFlag	填写YD_YOF_OptionExecute
	OrderRef	客户报单编号，详情请参考普通报单中关于OrderRef的说明
	Direction	填写YD_D_Sell
	OffsetFlag	填写行权对应仓位的OffsetFlag。 对于上期所和能源所，可以填写YD_OF_CloseToday或者YD_OF_CloseYesterday 对于其他交易所，填写YD_OF_Close
	OrderType	填写YD_ODT_Limit
	HedgeFlag	投保标志，期货交易所与股票期权交易所的定义不同。 期货交易所如下： YD_HF_Speculation=1：投机 YD_HF_Hedge=3：套保 股票期权交易所如下： YD_HF_Normal=1：普通 YD_HF_Covered=3：备兑
	OrderVolume	指定行权数量
YDInstrument		指定待行权的期权合约指针
YDAccount		填写NULL，表示使用当前API登陆账户

中金所不支持行权指令，在最后交易日中金所会自动行权所有实值期权，如果不想参与行权，请于最后交易日前平仓。

对于上交所和深交所，请参考以下方法使用insertOptionExecTogetherOrder或者checkAndInsertOptionExecTogetherOrder指令向交易所发起合并行权，并使用cancelOrder撤销合并行权请求。合并行权时，需要指定两个期权合约，他们必须满足以下条件：

- 期权合约标的证券应当相同，比如沪深300ETF期权合约与上证50ETF期权合约不能合并申报
- 认沽期权行权价必须高于认购期权且必须是当日到期的期权合约

- 合约单位必须相等。标的分红等会导致ETF期权的合约进行调整，会引起标准合约与非标准合约（合约代码非“M”的合约）的合约单位不同，而这类非标准合约与标准合约不能进行合并申报
- 认购和认沽期权的行权数量不得超过投资者持有的相应合约权利仓净头寸。净头寸以当日到期组合策略解除后的净头寸为准。若某次申报数量超过当前可用的行权额度，该笔申报全部无效。

参数	字段	说明
YDInputOrder	YDOrderFlag	填写YD_YOF_OptionExecuteTogether
	OrderRef	客户报单编号，详情请参考普通报单中关于OrderRef的说明
	OrderVolume	指定合并行权的数量。
YDInstrument	pInstrument	指定待合并行权的期权合约指针
	pInstrument2	指定待合并行权的期权合约指针
YDAccount		填写NULL，表示使用当前API登陆账户

6.1.11. 放弃期权行权

对于上期所、能源所、郑商所，可以参考以下说明使用insertOrder向交易所发起放弃行权申请，使用cancelOrder撤销放弃行权申请。放弃行权指令必须在美式期权最后交易日或者欧式期权的行权日发起。在上述交易日，若没有申请放弃行权，则实值期权会被交易所自动执行。

为便于编写程序，易达为是否支持使用YD_YOF_OptionAbandonExecute放弃期权行权提供了参数化表示，使用YDExchange.OptionAbandonExecutionSupport来表示对期权行权不同程度的支持：

- 0：不支持。目前中金所、上交所、深交所、大商所、广期所属于这个分类。中金所、上交所、深交所不支持放弃期权行权指令，大商所、广期所不支持使用YD_YOF_OptionAbandonExecute放弃期权行权，但是可以使用YD_YOF_Mark放弃期权行权，具体方法请参考下文。
- 1：支持但不包含风控，所谓不风控就是不做钱仓检查和冻结。目前没有交易所属于这个分类。
- 2：支持且包含风控，所谓风控就是做钱仓检查和冻结。目前上期所、能源所、郑商所属于这个分类。

下面介绍放弃期权行权的参数填写方法。

参数	字段	说明
YDInputOrder	YDOrderFlag	填写YD_YOF_OptionAbandonExecute
	OrderRef	客户报单编号，详情请参考普通报单中关于OrderRef的说明
	Direction	填写YD_D_Sell
	OffsetFlag	填写行权对应仓位的OffsetFlag。 对于上期所和能源所，可以填写YD_OF_CloseToday or YD_OF_CloseYesterday 对于其他交易所，填写YD_OF_Close

参数	字段	说明
	OrderType	填写YD_ODT_Limit
	HedgeFlag	投保标志，期货交易所与股票期权交易所的定义不同。 期货交易所如下： YD_HF_Speculation=1：投机 YD_HF_Hedge=3：套保 股票期权交易所如下： YD_HF_Normal=1：普通 YD_HF_Covered=3：备兑
	OrderVolume	指定行权数量
YDInstrument		指定待行权的期权合约指针
YDAccount		填写NULL，表示使用当前API登陆账户

对于大商所和广期所，可以参考以下说明使用insertOrder向交易所发起放弃行权申请，使用cancelOrder撤销放弃行权申请。放弃行权指令必须在行权日发起。放弃行权指令必须在美式期权最后交易日或者欧式期权的行权日发起。在上述交易日，若没有申请放弃行权，则期权会被自动执行。

参数	字段	说明
YDInputOrder	YDOrderFlag	填写YD_YOF_Mark
	OrderRef	客户报单编号 回报中的OrderRef始终为-1，和请求中的OrderRef无法对应
	Direction	YD_D_Buy：请求到期日自动行权 YD_D_Sell：撤销对应的请求
	OrderType	填写YD_ODT_OptionAbandonExecuteMark
	HedgeFlag	投保标志 YD_HF_Speculation=1：投机 YD_HF_Hedge=3：套保
YDInstrument		指定待放弃自动行权的期权合约指针
YDAccount		填写NULL，表示使用当前API登陆账户

对于上交所和深交所，并没有对应的放弃行权的指令，如果美式期权最后交易日或者欧式期权的行权日当日没有提出行权指令，则默认放弃行权。

对于中金所，并不支持放弃行权指令，如果不想参与行权，请于最后交易日前平仓。

6.1.12. 不支持业务

易达的目标是在保持性能的情况下尽可能支持投资者所需要的业务。基于性能和使用易达产品的投资者的实际业务需求考虑，易达柜台尚未支持以下业务。

交易所	不支持业务
上期所、能源所	结算价交易TAS指令

如果您对上述业务有实际需求的，请通过期货公司联系我们，假若该需求确实存在普遍性的，我们会努力满足您的业务需要。

6.2. 交易限制

6.2.1. 交易权限

出于风险管理和投资者适当性的需要，经纪商有在投资者、交易所、产品和合约层面设置交易权限的需求；投资者也有主动设置部分合约交易权限的风控需求。因此，易达提供了四个交易权限设置来源以满足不同的风控设置需求：

- 管理员设置的永久权限：设置后永久有效。只有管理员能设置该来源，投资者无法设置。
- 投资者设置的永久权限：设置后永久有效。管理员和投资者都可设置。
- 管理员设置的临时权限：设置后当前交易日内有效。只有管理员能设置该来源，投资者无法设置。目前由事后风控触发规则后设置，以确保事后风控规则只有在当前交易日有效。
- 投资者设置的临时权限：设置后当前交易日内有效。管理员和投资者都可设置。

> 无论合约的交易权限如何，撤单是始终可以执行的。

虽然易达允许在交易权限设置源分别设置，但投资者在具体合约上的权限是汇总上述四个来源后的结果。投资者设置交易权限的方法如下所示：

```
virtual bool setTradingRight(const YDAccount *pAccount,const YDInstrument *pInstrument,const YDProduct *pProduct,const YDExchange *pExchange, int tradingRight,int requestID=0,int tradingRightSource=YD_TRS_AdminPermanent)
```

上述方法中各参数的说明如下：

参数	说明
pAccount	指向账户的指针。当前投资者的账户指针可以通过getMyAccount获取。
pInstrument	合约指针，表示设置该合约的权限。此时请设置pProduct和pExchange为NULL
pProduct	产品指针，表示设置该产品的权限。此时请设置pInstrument和pExchange为NULL

参数	说明
pExchange	交易所指针，表示设置该交易所的权限。此时请设置pProduct和pInstrument为NULL
tradingRight	需要设置的权限。可选项为： YD_TR_Allow=0：允许所有交易。最为宽松的设置。 YD_TR_CloseOnly=1：仅允许平仓 YD_TR_Forbidden=2：不允许任何交易。最为严格的设置。
requestID	用于区分不同的设置请求，通常设置0即可。可以在notifyResponse中接收到请求的requestID。
tradingRightSource	交易权限设置来源 YD_TRS_AdminPermanent=0：管理员设置的永久权限 YD_TRS_UserPermanent=1：投资者设置的永久权限 YD_TRS_AdminTemp=2：管理员设置的临时权限 YD_TRS_UserTemp=3：投资者设置的临时权限

若需获取投资者在具体合约上的权限取决于投资者、该合约所属交易所、该合约所属产品以及该合约上最严格的设置，我们通过以下伪代码表述：

```
int getInstrumentTradingRight(YDApi *api, YDInstrument *instrument)
{
    accountInfo=api->getMyAccount();
    accountExchangeInfo=api->getAccountExchangeInfo(instrument->m_pExchange);
    accountProductInfo=api->getAccountProductInfo(instrument->m_pProduct);
    accountInstrumentInfo=api->getAccountInstrumentInfo(instrument);

    accountInfo->TradingRight = max(
        accountInfo->TradingRightFromSource[YD_TRS_AdminPermanent],
        accountInfo->TradingRightFromSource[YD_TRS_UserPermanent],
        accountInfo->TradingRightFromSource[YD_TRS_AdminTemp],
        accountInfo->TradingRightFromSource[YD_TRS_UserTemp]
    );

    accountExchangeInfo->TradingRight = max(
        accountExchangeInfo->TradingRightFromSource[YD_TRS_AdminPermanent],
        accountExchangeInfo->TradingRightFromSource[YD_TRS_UserPermanent],
        accountExchangeInfo->TradingRightFromSource[YD_TRS_AdminTemp],
        accountExchangeInfo->TradingRightFromSource[YD_TRS_UserTemp]
    );

    accountProductInfo->TradingRight = max(
        accountProductInfo->TradingRightFromSource[YD_TRS_AdminPermanent],
```

```
        accountProductInfo->TradingRightFromSource[YD_TRS_UserPermanent],
        accountProductInfo->TradingRightFromSource[YD_TRS_AdminTemp],
        accountProductInfo->TradingRightFromSource[YD_TRS_UserTemp]
    );

    accountInstrumentInfo->TradingRight = max(
        accountInstrumentInfo->TradingRightFromSource[YD_TRS_AdminPermanent],
        accountInstrumentInfo->TradingRightFromSource[YD_TRS_UserPermanent],
        accountInstrumentInfo->TradingRightFromSource[YD_TRS_AdminTemp],
        accountInstrumentInfo->TradingRightFromSource[YD_TRS_UserTemp]
    );

    return max(
        accountInfo->TradingRight,
        accountExchangeInfo->TradingRight,
        accountProductInfo->TradingRight,
        accountInstrumentInfo->TradingRight
    );
}
```

盘中若交易权限发生改变，通常是由于管理员主动设置或者事后风控触发，可以通过以下回调函数获得权限变更通知。收到任何下面三个回调中的一个时，都应该使用上面的伪代码函数getInstrumentTradingRight获取您关心的合约的交易权限：

```
virtual void notifyAccount(const YDAccount *pAccount)
virtual void notifyAccountExchangeInfo(const YDAccountExchangeInfo *pAccountExchangeInfo)
virtual void notifyAccountProductInfo(const YDAccountProductInfo *pAccountProductInfo)
virtual void notifyAccountInstrumentInfo(const YDAccountInstrumentInfo *pAccountInstrumentInfo)
```

如果尝试在没有交易权限的合约上开平仓，会收到错单回报notifyOrder或者错误报价回报notifyQuote，其中的ErrorNo为YD_ERROR_NoTradingRight=10。

6.2.2. 报单量限制

默认情况下，易达柜台并不控制投资者的报单数量，但是考虑到易达柜台的内存数据库存在上限，如果数据库中某个表达到上限，整个易达柜台就会处于不可用状态。为了避免某些投资者异常交易导致的交易量暴增而影响到柜台的其他客户，经纪商可以为每个投资者设置报单量限制，报单量控制是在资金账户层面的，如果投资者建立了多个连接，所有连接的交易数量是汇总计算的。只有普通报单发送到交易所并收到交易所回报会增加投资者的报单量，报价、撤单、衍生报单、柜台返回的错误报单都不会计入报单量。

> 当前版本中，易达柜台内存数据库报单、成交的上限为1677万笔，报价的上限为838万笔，如业务发生变化，正常业务需要更大的空间，易达随时可以扩容。

报单量限制数值可以在YDAccount.MaxOrderCount中查到，如果该值为-1，则表示不限制，默认为不限制。

对于使用ydExtendedApi的投资者，可以在YDExtendedAccount.UsedOrderCount查询到当前账户的总报单数。如果投资者使用该资金账户建立了多个连接，YDExtendedAccount.UserdOrderCount包含了所有连接的报单数。

在投资者达到该账户的报单量上限后的下一张报单会收到notifyOrder返回的YD_ERROR_TooManyOrders=36错误。为保护柜台表空间不被超量非正常报单挤满，如果该投资者继续报单将直接被柜台丢弃，不会有任何错误回报返回。请投资者务必检查YD_ERROR_TooManyOrders=36错误并在收到该错误后停止或者调整报单逻辑，如有疑问请联系经纪商核实。

如果柜台内存数据库资源耗尽，投资者的任意报单都会收到notifyOrder返回的YD_ERROR_MemoryExceed=7错误。

6.2.3. 登陆数限制

默认情况下，易达柜台不会限制资金账户的登陆连接个数，投资者可以使用同一个账户无限制的登陆易达柜台。但是，过多的连接会造成TCP下行线程始终忙于下发下行流水，这对易达柜台的高效运行是有害的。为了避免连接数过多而影响柜台的性能，经纪商可以为每个投资者设置登陆数限制。

登陆数总数可以在YDAccount.MaxLoginCount中查到，如果该值为-1，则表示不限制，默认为不限制。当前登陆数可以在YDAccount.LoginCount中查到。

在投资者达到该账户的登陆数上限后，继续登陆会在notifyLogin中返回YD_ERROR_TooManyRequests=20错误。

如果了解更多多连接的情况，请参阅多连接相关内容。

6.2.4. 自成交检查

为避免因投资者自成交被交易所限制交易，易达柜台提供了自成交检查功能。易达的自成交检查功能做了性能优化，因此不用因为担心性能问题而关闭自成交检查。

该功能默认是打开的，如果投资者在策略程序中已经做了检查，可以请经纪商帮助关闭该账户的自成交检查功能。投资者可以检查YDAccount.AccountFlag中是否被设置了YD_AF_DisableSelfTradeCheck标志，如果被设置则表示该账户的自成交检查是关闭的。

交易所规定只有限价报单与已处于挂单状态的限价报单价格重叠才会构成自成交。因此，市价单、FAK单、FOK单不会进入易达自成交检查逻辑，而如果限价单和现有挂单价格的买卖价格不重叠，也不会被认定为自成交。例如，若某合约当前有卖单价格10元，若此时想要报入11元的买单，那么就会被系统判定为自成交，而如果报入9元的买单，则不会被判定为自成交。

> 如果在中金所交易时遇到了"1138 订单触发自成交"的错误，说明所在经纪商已经向中金所申请开通了"自成交防范功能"，该功能会检查同一交易编码下的FAK/FOK单与限价单是否发生价格重叠，若重叠则会被交易所拦截。自成交防范功能是中金所向经纪商提供的一种保护功能，启用后对该期货公司所有交易编码生效，不可以在投资者维度单独设置。虽然自成交防范功能拦截了FAK/FOK的自成交行为，但不代表FAK/FOK单引起的自成交会被中金所监管部门判定为自成交。在《中国金融期货交易所异常交易管理办法》股指期权有关监管标准及处理程序中明确指出，在统计客户自成交、频繁报撤单和大额报撤单次数时，因即时全部成交或撤销限价指令、即时成交剩余撤销限价指令、市价指令形成的自成交行为不计入自成交次数统计。

虽然某些交易所给予投资者一定的自成交豁免额，但是考虑到自成交检查是基于投资者交易编码的，如果同一个投资者的交易编码在多家经纪商交易造成自成交的，也会被交易所计算在自成交范围内，而此类自成交控制是较为困难的。出于保护投资者的角度考虑，为了把豁免额留给不太可控的同交易编码跨经纪商自成交，易达柜台不允许任何潜在的自成交报单，所有被系统检测为可能构成自成交的都会被柜台直接拒绝。

任何被判定为自成交的报单，都会通过notifyOrder返回的YD_ERROR_PossibleSelfTrade=25错误。

6.2.5. 报单号单调递增检查

在过去的API版本中，报单和报价等上行指令中的OrderRef是没有递增要求的，投资者可以根据需要随意设置OrderRef。易达柜台在生产中曾经发生过因为网络原因，柜台UDP接口收到大量重复报单，导致投资者交易受到了影响。

为了避免这种问题的发生，我们建议投资者使用非0报单组进行报单，利用其检查功能规避重复报单。详情请参见报单组。

> 在过去版本的API中，为解决上述问题，易达为每个在管理端新开立的资金账户默认打开了报单号检查功能。柜台将检查打开该功能的投资者的OrderRef是否单调递增的，若收到了OrderRef非单调递增的报单，则将返回ErrorNo为YD_ERROR_InvalidOrderRef=78的错误回报。
>
> 报单号单调递增检查的范围仅限于API实例的范围内，即同一个API实例发出的报单要求单调递增，但是相同账户创建的多个API实例之间的报单的OrderRef没有任何限制，因此，多连接场景下，使用OrderRef后几位作为SessionID的做法仍然是生效的。裸协议报单是作为一个单独的源进行检查的，不会和API的UDP报单相冲突。若柜台发生了重启（如日盘启动）而客户端因为开启了客户端高可用而没有重启的，报单号可以重头开始。
>
> 投资者可以检查YDAccount.AccountFlag是否设置了YD_AF_OrderRefCheck标志了解该功能的启用情况，若设置了该标志则表示报单号单调递增检查功能开启。
>
> 为了避免不必要的重复检查，我们建议对于已经使用非0报单组进行报单的投资者关闭上述报单号检查功能。若投资者继续使用编号为0的报单组进行报单同时不希望报单号检查影响现有OrderRef的编码逻辑，也可以请经纪商关闭账号上的报单检查开关。

6.3. 查询交易信息

ydApi并不保存流水，因此下列查询功能只能在ydExtendedApi中使用。

查询语句是慢速调用，不应在交易的线程中调用，也尽可能不要频繁调用查询语句，以免造成系统卡顿。

6.3.1. 查询报单

```
/// getOrder by orderSysID can only be used for orders have been accepted by exchange
virtual const YDExtendedOrder *getOrder(int orderSysID,const YDExchange *pExchange,int YDOrderFlag=YD_YOF_Normal)
virtual const YDExtendedOrder *getOrder(long long longOrderSysID,const YDExchange *pExchange,int YDOrderFlag=YD_YOF_Normal)
```

上面的方法可以用来查询**已经收到交易所回报的报单**或者报价衍生报单，所有通过insertOrder系列报单方法报入的报单都可以使用本函数查询到，默认情况下查询普通报单，可以通过设置YDOrderFlag查询其他类型的报单。

```
/// getOrder by orderRef can only be used for orders using checkAndInsertOrder
virtual const YDExtendedOrder *getOrder(int orderRef,unsigned orderGroupID=0,const YDAccount *pAccount=NULL)=0;

/// getQuoteDerivedOrder can only be used for orders derived oder by using checkAndInsertQuote
virtual const YDExtendedOrder *getQuoteDerivedOrder(int orderRef,int direction,unsigned orderGroupID=0,const YDAccount *pAccount=NULL)=0;
```

对于使用ydExtendedApi的checkAndInsertOrder报入的报单可以使用上述两个方法获取，只要checkAndInsertOrder和checkAndInsertQuote调用完成，就能通过这两个方法查询到，而无需收到交易所报单回报。其中getOrder可以查询到除了报价衍生报单之外的报单，而getQuoteDerivedOrder可以查询报价衍生报单，相比getOrder增加了报单方向direction的参数用于区分双边报价中的买卖衍生报单。

```
/// orders must have spaces of count, return real number of orders(may be greater than count). Only partial will be set if no enough space. Only orders accepted by exchange can be found in this function
virtual unsigned findOrders(const YDOrderFilter *pFilter,unsigned count,const YDExtendedOrder *orders[])=0;

/// User should call destroy method of return object to free memory after using following method
virtual YDQueryResult<YDExtendedOrder> *findOrders(const YDOrderFilter *pFilter)=0;
```

易达提供了上述两种不同的批量查询**已经收到交易所回报的报单**的方法，他们的查询条件的使用方法是相同的，判断某报单是否符合查询条件的伪代码逻辑如下所示：

```
if YDOrder.OrderSysID < 0
    return false

if YDOrder.YDOrderFlag not in YDOrderFilter.YDOrderFlags
    return false

if YDOrderFilter.StartTime >= 0 and YDOrder.InsertTime >
YDOrderFilter.StartTime
    return false

if YDOrderFilter.EndTime >= 0 and YDOrder.InsertTime <
YDOrderFilter.EndTime
    return false

if YDOrderFilter.pExchange != NULL and YDOrderFilter.pExchange !=
YDOrder.pExchange
    return false

if YDOrder.YDOrderFlag ==  YD_YOF_CombPosition
    if YDOrderFilter.pCombPositionDef != NULL and
YDOrderFilter.pCombPositionDef != YDOrder.pCombPositionDef
        return false
else
    if YDOrderFilter.pInstrument != NULL and YDOrderFilter.pInstrument !=
YDOrder.pInstrument
        return false
    if YDOrderFilter.pProduct != NULL and YDOrderFilter.pProduct !=
YDOrder.pProduct
        return false
return true
```

各个字段填写说明如下：

参数	字段	说明
YDOrderFilter	StartTime	以TimeID形式表示的起始时间，-1代表无限制。请参考以下例子： 夜盘21点整：3600*(21-17)=14400 日盘9点整：3600*(24+9-17)=57600 周一日盘9点整：3600*(24+9-17)=5760 TimeID和标准时间的转换程序参见ydUtil.h中的string2TimeID和timeID2String

参数	字段	说明
	EndTime	以TimeID形式表示的结束时间，-1代表无限制。请参考以下例子： 夜盘21点整：3600*(21-17)=14400 日盘9点整：3600*(24+9-17)=57600 周一日盘9点整：3600*(24+9-17)=5760 TimeID和标准时间的转换程序参见ydUtil.h中的string2TimeID和timeID2String
	YDOrderFlags	设置要查询的YDOrderFlag的标志位，可以设置多个。例如，要查询普通报单和组合报单，那么可以使用以下表达式设置： 1<<YD_YOF_Normal | 1<<YD_YOF_CombPosition
	pCombPositionDef	组合持仓定义。对于YD_YOF_CombPosition的报单只检查本参数和pExchange。如设置为NULL则不限制。
	pInstrument	合约指针，对于非YD_YOF_CombPosition的报单生效。如设置为NULL则不限制。
	pProduct	产品指针，对于非YD_YOF_CombPosition的报单生效。如设置为NULL则不限制。
	pExchange	交易所指针。如设置为NULL则不限制。
	pAccount	投资者请始终设置为NULL

第一种方法需要投资者事先分配固定长度的YDExtendedOrder指针数组，如果预分配的长度不足以容纳的，只会填入预分配数组长度的报单。无论预分配长度是否足够，本方法的返回值都是返回满足查询条件的报单的总数，投资者可以利用这个特点，可以调用findOrders(pFilter, 0, NULL)来快速获取满足条件的报单的总数。

- 本方法的好处是在报单不多而且预分配数组长度足够时，可以重复利用预分配的指针数组，而不用每次查询都分配
- 本方法的缺点是如果报单较多时，可能需要两次调用（第一次获取总报单数，第二次分配可容纳所有报单的数组后再次调用）才能完整获取所有满足条件的报单

第二种方法可以帮助投资者分配能容纳所有满足条件的报单的空间，但是需要投资者在使用完成后主动调用YDQueryResult.destory()销毁分配的空间。相比起第一种方法：

- 本方法的好处是调用一次就返回所有的报单
- 本方法的缺点是需要投资者主动释放由本方法分配的空间，而且每次调用都会分配新的空间，频繁的分配与释放对缓存非常不友好。

6.3.2. 查询成交

```
/// trades must have spaces of count, return real number of trades(may be greater than count). Only partial will be set if no enough space
virtual unsigned findTrades(const YDTradeFilter *pFilter,unsigned count,const YDExtendedTrade *trades[])=0;

/// User should call destroy method of return object to free memory after using following method
virtual YDQueryResult<YDExtendedTrade> *findTrades(const YDTradeFilter *pFilter)=0;
```

批量查询成交的方法与查询报单类似，具体内容可以参考查询报单的相关内容。判断某成交是否符合查询条件的伪代码逻辑如下所示：

```
if YDTradeFilter.StartTime >= 0 and YDTrade.InsertTime > YDTradeFilter.StartTime
    return false

if YDTradeFilter.EndTime >= 0 and YDTrade.InsertTime < YDTradeFilter.EndTime
    return false

if YDTradeFilter.pInstrument != NULL and YDTradeFilter.pInstrument != YDTrade.pInstrument
    return false

if YDTradeFilter.pProduct != NULL and YDTradeFilter.pProduct != YDTrade.pProduct
    return false

if YDTradeFilter.pExchange != NULL and YDTradeFilter.pExchange != YDTrade.pExchange
    return false

return true
```

各个字段填写说明如下：

参数	字段	说明

参数	字段	说明
YDTradeFilter	StartTime	以TimeID形式表示的起始时间，-1代表无限制。请参考以下例子： 夜盘21点整：3600*(21-17)=14400 日盘9点整：3600*(24+9-17)=57600 周一日盘9点整：3600*(24+9-17)=5760 TimeID和标准时间的转换程序参见ydUtil.h中的string2TimeID和timeID2String
	EndTime	以TimeID形式表示的结束时间，-1代表无限制。请参考以下例子： 夜盘21点整：3600*(21-17)=14400 日盘9点整：3600*(24+9-17)=57600 周一日盘9点整：3600*(24+9-17)=5760 TimeID和标准时间的转换程序参见ydUtil.h中的string2TimeID和timeID2String
	pInstrument	合约指针。如设置为NULL则不限制。
	pProduct	产品指针。如设置为NULL则不限制。
	pExchange	交易所指针。如设置为NULL则不限制。
	pAccount	投资者请始终设置为NULL

6.3.3. 查询报价

```
/// getQuote by quoteSysID can only be used for quotes have been accepted by exchange
virtual const YDExtendedQuote *getQuote(int quoteSysID,const YDExchange *pExchange)
virtual const YDExtendedQuote *getQuote(long long longQuoteSysID,const YDExchange *pExchange)
```

上面的方法可以用来查询**已经收到交易所回报的报价**，所有通过insertQuote系列报价方法报入的报价都可以使用本函数查询到。

```
/// getQuote by orderRef can only be used for quotes using checkAndInsertQuote
virtual const YDExtendedQuote *getQuote(int orderRef,unsigned orderGroupID=0,const YDAccount *pAccount=NULL)=0;
```

对于使用ydExtendedApi的checkAndInsertQuote报入的报价可以使用上述方法获取，只要checkAndInsertQuote调用完成，就能通过这两个方法查询到，而无需收到交易所报单回报。

```
/// quotes must have spaces of count, return real number of quotes(may be greater than count). Only partial will be set if no enough space. Only quotes accepted by exchange can be found in this function
virtual unsigned findQuotes(const YDQuoteFilter *pFilter,unsigned count,const YDExtendedQuote *quotes[])=0;

/// User should call destroy method of return object to free memory after using following method
virtual YDQueryResult<YDExtendedQuote> *findQuotes(const YDQuoteFilter *pFilter)=0;
```

批量查询报价的方法与查询报单类似，具体内容可以参考查询报单的相关内容。判断某报价是否符合查询条件的伪代码逻辑如下所示：

```
if YDQuote.OrderSysID < 0
    return false

if YDQuoteFilter.StartTime >= 0 and (any DerivedOrder of YDQuote).InsertTime > YDQuoteFilter.StartTime
    return false

if YDQuoteFilter.EndTime >= 0 and (any DerivedOrder of YDQuote).InsertTime < YDQuoteFilter.EndTime
    return false

if YDQuoteFilter.pInstrument != NULL and YDQuoteFilter.pInstrument != YDQuote.pInstrument
    return false

if YDQuoteFilter.pProduct != NULL and YDQuoteFilter.pProduct != YDQuote.pProduct
    return false

if YDQuoteFilter.pExchange != NULL and YDQuoteFilter.pExchange != YDQuote.pExchange
    return false

return true
```

各个字段填写说明如下：

参数	字段	说明

参数	字段	说明
YDTradeFilter	StartTime	以TimeID形式表示的起始时间，-1代表无限制。报价的任意衍生报单的时间大于开始时间就符合条件。 请参考以下例子： 夜盘21点整：3600*(21-17)=14400 日盘9点整：3600*(24+9-17)=57600 周一日盘9点整：3600*(24+9-17)=5760 TimeID和标准时间的转换程序参见ydUtil.h中的string2TimeID和timeID2String
	EndTime	以TimeID形式表示的结束时间，-1代表无限制。报价的任意衍生报单的时间小于结束时间就符合条件。 请参考以下例子： 夜盘21点整：3600*(21-17)=14400 日盘9点整：3600*(24+9-17)=57600 周一日盘9点整：3600*(24+9-17)=5760 TimeID和标准时间的转换程序参见ydUtil.h中的string2TimeID和timeID2String
	pInstrument	合约指针。如设置为NULL则不限制。
	pProduct	产品指针。如设置为NULL则不限制。
	pExchange	交易所指针。如设置为NULL则不限制。
	pAccount	投资者请始终设置为NULL

6.4. 交易阶段

易达支持发送汇总和明细两种交易阶段通告：

- 汇总交易阶段通告是始终发送的交易阶段通告。为了减少通讯量和对API的冲击，汇总交易阶段通告仅通知客户端交易所发布状态变化的时刻，若同一时刻有多个事件发生，只有第一个会发出，即相同时刻的事件只会推送第一个。
- 明细交易阶段通告是默认不发送的交易交易阶段通告。由于某些交易所的交易阶段通告是基于合约的，导致在同一时间会有大量的通告向所有投资者发送，如果此时有报单回报则会被拥塞。因此，请谨慎使用明细交易阶段通告。一旦本功能被打开，则该柜台上所有投资者都将收到明细交易阶段通告。可以通过SystemParam中的TradingSegmentDetail查看柜台是否启用该功能，详情参见系统参数。

6.4.1. 汇总交易阶段

汇总交易阶段通过以下回调函数返回。segmentTime表示从该交易日开始到此时间的秒数，其编码方式请参见报单回报中InsertTime的相关说明。

```
virtual void notifyTradingSegment(const YDExchange *pExchange, int segmentTime)
```

下面列出了目前各个交易所的交易状态变化时间供参考，若交易所有变化恕不另行通知，请以生产中接收到的实际时间为准。

交易所	日盘	夜盘
中金所	09:25:00 09:29:00 09:30:00 11:30:00 13:00:00 14:57:00 15:00:00 15:15:00	
能源所	09:00:00 10:15:00 10:30:00 11:30:00 13:30:00 15:00:00	20:55:00 20:59:00 21:00:00 23:00:00 01:00:00 02:30:00
上期所	08:55:00 08:59:00 09:00:00 10:15:00 10:30:00 11:30:00 13:30:00 15:00:00	20:55:00 20:59:00 21:00:00 23:00:00 01:00:00 02:30:00

交易所	日盘	夜盘
大商所	08:54:50 08:55:00 08:58:50 08:59:00 08:59:50 09:00:00 10:14:50 10:15:00 10:29:50 10:30:00 11:29:50 11:30:00 13:29:50 13:30:00 14:59:50 15:00:00	20:54:50 20:55:00 20:58:50 20:59:00 20:59:50 21:00:00 22:59:50 23:00:00

6.4.2. 明细交易阶段

明细交易阶段通过以下回调函数返回。交易阶段的返回维度各个交易所不尽相同，可能在交易所、产品或合约共三个维度上返回明细交易阶段，每次明细交易阶段通告只可能上述三个维度中的一个。

```
virtual void notifyTradingSegmentDetail(const YDTradingSegmentDetail *pTradingSegmentDetail)
```

返回的YDTradingSegmentDetail的字段说明如下。

字段	说明
ExchangeRef	交易所序号。返回交易所明细交易阶段时有意义，否则为-1。
ProductRef	产品序号。返回产品明细交易阶段时有意义，否则为-1。
InstrumentRef	合约序号。返回合约明细交易阶段时有意义，否则为-1。
m_pExchange	交易所指针。返回交易所明细交易阶段时有意义，否则为NULL。
m_pProduct	产品指针。返回产品明细交易阶段时有意义，否则为NULL。
m_pInstrument	合约指针。返回合约明细交易阶段时有意义，否则为NULL。
SegmentTime	从该交易日开始到此时间的秒数。其编码方式请参见报单回报中InsertTime的相关说明。

字段	说明
TradingStatus	交易阶段状态： YD_TS_NoTrading：非交易 YD_TS_Continuous：连续交易 YD_TS_Auction：集合竞价

6.5. 报单组

为了更好的支持投资者对区分报单和报单号单调递增检查的需求，我们在`1.280`版本中增加了报单组功能。投资者可以在YDInputOrder和YDInputQuote中设置OrderGroupID和GroupOrderRefControl使用报单组功能。

字段	说明
OrderGroupID	指定报单和报价所属的逻辑组，可以用于区分报单所属的策略、连接以及其他逻辑分组 投资者可以使用0-63号报单组。 为保兼容，0号报单组不检查OrderRef单调递增性，1-63号报单组将按设置检查OrderRef单调性
GroupOrderRefControl	指定投资者要求柜台对报单OrderRef的单调递增性检查方式： YD_GORF_Increase：OrderRef单调递增，前后两单OrderRef的间隔必须大于等于1 YD_GORF_IncreaseOne：OrderRef严格单调递增，前后两单OrderRef的间隔必须为1

在报单和报价上指定OrderGroupID和GroupOrderRefControl后，柜台将检查报单所属报单组的单调性。若检查失败，则直接通过notifyOrder返回ErrorNo=YD_ERROR_InvalidGroupOrderRef的错单，并在该错单的MaxOrderRef中返回报单中设置的OrderGroupID当前最大OrderRef。无论报单成功与失败，YDInputOrder和YDInputQuote上设置的OrderGroupID和GroupOrderRefControl都将通过YDOrder和YDQuote带回。

通过在每张报单上指定OrderGroupID和GroupOrderRefControl，给予了投资者最大的使用便利。对属于同一OrderGroupID的报单，投资者可以对第一张设置YD_GORF_Increase实现单调递增检查，第二张设置YD_GORF_IncreaseOne实现严格单调递增检查。

在首次登陆成功或者断线重连登陆成功的回调函数notifyLogin之后，柜台将通过以下回调函数返回每个OrderGroupID当前最大OrderRef。请务必记录该回调中的返回值，并在后续报单中使用符合递增规范的OrderRef进行报单。

```
virtual void notifyGroupMaxOrderRef(const int groupMaxOrderRef[])
```

对于使用ydExtendedApi的投资者，可以使用下列方法直接获取报单或者报价的下个OrderRef：

```
virtual int getNextOrderRef(unsigned orderGroupID=0,bool update=true)=0;
```

orderGroupID指定报单组编号。若orderGroupID设置为0，可以设置下个OrderRef的规则，则请参考多连接；若orderGroupID设置为1至63，那么下个OrderRef必定为当前最大OrderRef加1。

update指示获取下个OrderRef后，是否需要更新该报单组的最大报单编号。假设本次调用getNextOrderRef返回的下个OrderRef为n，若本次调用中update为true，那么下次调用getNextOrderRef返回的下个OrderRef为n+1；若本次调用中update为false，那么下次调用getNextOrderRef返回的下个OrderRef仍然为n。

6.6. 多连接

易达柜台默认支持同一个资金账号同时启动多个API实例连接柜台，而且默认情况下无连接数限制。经纪商可以在柜台端控制每个投资者的登陆连接个数上限，投资者可以在YDAccount的MaxLoginCount查询经纪商是否设置了连接上线，当没有限制时MaxLoginCount为-1。同时，YDAccount的LoginCount记录了当前该资金账户连接的总数。对于同一个资金账号的不同连接，MaxLoginCount和LoginCount总是相同的。

若投资者需要区分不同连接的报单，请优先使用报单组功能。

> 在过去版本的API中，易达为使用ydExtendedApi的投资者提供了一套利用OrderRef的低位编码SessionID的机制，但由于限制较多，我们不再推荐使用。为保持兼容性，当前版本中仍然为编号为0的报单组保留该机制，若投资者使用非0报单组则下述内容无效。
>
> 请注意，一旦决定使用该机制，请在所有连接中配置使用该机制，否则会导致潜在的OrderRef编码冲突。对于使用了本地风控报单的多连接投资者，请务必使用SessionID机制，否则所有连接发出的OrderRef都会冲突。
>
> 该连接号机制涉及以下函数：
>
> ```
> virtual bool setSessionOrderRefRule(unsigned sessionBitCount,unsigned sessionID)=0;
> virtual void getSessionOrderRefRule(unsigned *pSessionBitCount,unsigned *pSessionID)=0;
> ```
>
> 请**在任何报单和报价前**，通过setSessionOrderRefRule设置线程号以及全局线程号规则，setSessionOrderRefRule的各参数说明如下：
>
> - sessionBitCount设置了使用OrderRef中用于SessionID的末尾保留位数，保留位数决定了最大连接数和最大报单数，保留位数最多16位，这个保留位数必须在所有的连接中都相同
> - sessionID则标记了本连接的连接号，请从0开始为每个API实例的连接编号，请确保连接总数不超过sessionBitCount指定的个数范围
>
>   举例来说，假设某连接设置sessionBitCount为8，SessionID为3，那么OrderRef中有8位用于连接号，剩下的24位（32-8=24位）用于真正的报单编号，因此，在这种设置模式下，可以有256（2^8）个连接号，每个连接可以有16,777,216（2^24）张报单，同时，本API实例的连接号为3。该连接的第一个OrderRef为259（0b100000000+0b11=0b100000011=259）。

> 在设置完成后，即可以使用getSessionOrderRefRule获取设置的连接号编码规则，以及使用getNextOrderRef获取下一个报单编号。

6.7. 裸协议

若使用`1.280`及以上版本API，请严格按照以下裸协议规范报单和接收回报，否则将会被认为错单。若使用`1.188`及以下版本API，请参考对应版本的裸协议规范。

6.7.1. 上行报文

从`1.52`版本开始，易达开放了报单和撤单报文，并在`1.280`版本中增加开放了报价、撤报价以及各种回报报文。投资者可以向易达柜台发送自组的UDP报文进行交易。由于裸协议报单是指用户自己负责数据包的组装和发送，其性能直接取决于用户的实现，和易达API没有关系。无论是通过裸协议报单还是通过易达API的UDP方式报单，在柜台端的穿透性能是没有区别的，因此，与API的性能差异仅仅在于客户端发送端的性能。

易达的API经过长时间的迭代，在我们实验室中（在X10/25和X2522网卡上）测得的性能已经令人满意，我们认为ydAPI发送性能已经能达到普通FPGA的水准。如果客户尝试自己实现裸协议报单，那么请在实现完成并与我们的API的报单性能进行比较后，选择较快的一种作为您生产报单的方式。

API提供了时间戳函数getYDNanoTimestamp，该函数精度高速度快，可以用来测试API的报单速度。具体方法为在insertOrder的前后调用getYDNanoTimestamp并将结果相减，得到的是从开始发送到报单的第一个字节出现在光纤上的时间差。

裸协议报撤单只能通过UDP报送，UDP端口号可以在SystemParam中获取，详情参见系统参数。源端口号可以任意，易达不检查源端口号。

6.7.1.1. 使用准备

投资者需要得到经纪商的批准后才能使用裸协议，使用前请与经纪商联系开通。投资者可以检查YDAccount.AccountFlag是否设置了YD_AF_BareProtocol以确认是否已经开通了裸协议。

UDP报文头需要通过调用getClientPacketHeader获取，报文头中包含了账户、密钥等信息，因此不可修改报文头中任何内容。只能在notifyFinishInit及其后续步骤中获取报文头。一旦成功获得了报文头，本次柜台启动期间是不变的，如果柜台发生了重启，则需重新获取。

```
virtual int getClientPacketHeader(YDPacketType type,unsigned char
*pHeader,int len)=0;
```

YDPacketType可以是YD_CLIENT_PACKET_INSERT_ORDER、YD_CLIENT_PACKET_CANCEL_ORDER、YD_CLIENT_PACKET_INSERT_QUOTE、YD_CLIENT_PACKET_CANCEL_QUOTE，分别用于获取报单、撤单、报价、撤报价的报文头。header和len是用户预先创建的存储区域的头指针以及长度，用于容纳api生成的头部数据，len应大于对应类型的头部的长度，目前各业务报文头长度都是16。

返回值表示实际返回的报文头长度，如果返回0，表示获取报文头失败，通常是因为预先创建的存储区域长度不够，调用时间点早于notifyFinishInit或者没有开通裸协议报单功能YD_AF_BareProtocol。

6.7.1.2. 报单报文

以下为报单时的报文结构，等同于调用insertOrder。

地址偏移	长度(字节)	字段类型	注释
0	16	整数	报文头。
16	4	整数（little-endian）	合约序号。从YDInstrument的InstrumentRef中获取。
20	1	整数	买卖方向(0：买，1：卖)
21	1	整数	开平标志(0：开仓，1：平仓，3：平今，4：平昨)
22	1	整数	投保标志(1：投机，2：套利，3：套保)
23	1	整数	席位选择方法 (0：YD_CS_Any, 1：YD_CS_Fixed, 2：YD_CS_Prefered)
24	8	IEEE 754双精度浮点类型（little-endian）	价格
32	4	整数（little-endian）	报单数量
36	4	整数（little-endian）	报单引用
40	1	整数	报单类型(0：限价单，1：FAK, 2:市价单，3:FOK)
41	1	整数	填0
42	1	整数	席位连接编号
43	5	整数	填0
48	1	无符号整数	报单组ID
49	1	整数	报单组OrderRef控制方式，详情参见报单组 0：OrderRef单调递增，前后两单OrderRef的间隔必须大于等于1 1：OrderRef严格单调递增，前后两单OrderRef的间隔必须为1
50	1	整数	触发单类型 0：无触发条件 1：止盈触发 2：止损触发

地址偏移	长度(字节)	字段类型	注释
51	13	整数	填0
64	8	IEEE 754双精度浮点类型（little-endian）	触发单触发价格

6.7.1.3. 撤单报文

以下为撤单时的报文结构，等同于调用cancelOrder。目前支持用全精度的交易所报单编号、交易所报单编号、委托编号OrderRef三种撤单方式，具体请参考普通撤单。

地址偏移	长度(字节)	字段类型	注释
0	16	整数	报文头。
16	4	整数（little-endian）	欲撤单的交易所报单编号或者OrderRef。
20	1	整数	交易所序号。从YDExchange的ExchangeRef中获取。
21	1	整数	席位选择方法 (0：YD_CS_Any, 1：YD_CS_Fixed, 2：YD_CS_Prefered)
22	1	整数	席位连接序号
23	1	整数	填0
24	1	无符号整数	报单组ID
25	7	整数	填0
32	8	整数（little-endian）	全精度的交易所报单编号

6.7.1.4. 报价报文

以下为报单时的报文结构，等同于调用insertQuote。

地址偏移	长度(字节)	字段类型	注释
0	16	整数	报文头。

地址偏移	长度(字节)	字段类型	注释
20	4	整数（little-endian）	合约序号。从YDInstrument的InstrumentRef中获取。
24	1	整数	买开平标志
25	1	整数	买投保标志
26	1	整数	卖开平标志
27	1	整数	卖投保标志
28	8	IEEE 754双精度浮点类型（little-endian）	买价格
36	8	IEEE 754双精度浮点类型（little-endian）	卖价格
44	4	整数（little-endian）	买量
48	4	整数（little-endian）	卖量
52	4	整数（little-endian）	报单引用，即OrderRef
56	1	整数	席位选择方法 (0：YD_CS_Any, 1：YD_CS_Fixed, 2：YD_CS_Prefered)
57	1	整数	席位连接编号
58	1	整数	填0
59	1	整数	报价标志 YD_YQF_ResponseOfRFQ：自动填写询价号表示应价，支持的有上期所、能源所、大商所、广期所、郑商所，其他交易所无询价号
60	1	无符号整数	报单组ID

地址偏移	长度(字节)	字段类型	注释
61	1	整数	报单组OrderRef控制方式，详情参见报单组 0：OrderRef单调递增，前后两单OrderRef的间隔必须大于等于1 1：OrderRef严格单调递增，前后两单OrderRef的间隔必须为1
62	6	整数（little-endian）	填0

6.7.1.5. 撤报价报文

以下为撤报价时的报文结构，等同于调用cancelQuote。目前支持用全精度的交易所报单编号、交易所报单编号、委托编号OrderRef三种撤报价方式，具体请参考普通撤报价。

地址偏移	长度(字节)	字段类型	注释
0	16	整数	报文头。
16	4	整数（little-endian）	欲撤报价的交易所报价编号或者OrderRef。
20	1	整数	交易所序号。从YDExchange的ExchangeRef中获取。
21	1	整数	席位选择方法 (0：YD_CS_Any, 1：YD_CS_Fixed, 2：YD_CS_Prefered)
22	1	整数	席位连接序号
23	1	无符号整数	报单组ID
24	8	整数（little-endian）	全精度的交易所报价编号

6.7.2. 下行报文

易达的下行报文通过TCP发送，在能高效解决UDP可靠送达的问题之前，我们不会考虑增加UDP下行报文，需要解析下行报文的投资者需要自行旁路监听TCP报文段，并正确处理可能发生的重发、黏连等问题。

6.7.2.1. 报文头

易达的下行报文使用相同的头部结构，收到下行报文后，首先需要解析报文头中的报文类型以区分使用何种报文结构解析后续数据。报文头的结构如下表所示：

地址偏移	长度(字节)	字段类型	注释
0	2	整数（little-endian）	报文长度（包含本报文头）
2	2		非公开字段
4	4	整数（little-endian）	报文类型
8	4		非公开字段
12	4	整数（little-endian）	投资者账户序号

上述报文头中可能的报文类型如下表所示：

报文类型	说明
0	心跳
34	报单回报
35	成交回报
37	询价
40	报价回报
43	撤单或撤报价失败回报

6.7.2.2. 报单回报报文

由于大量业务复用了报单回报的报文，因此请只关注报单标志为0的报单回报，非0的报单回报应忽略。

地址偏移	长度(字节)	字段类型	注释
0	16		报文头
16	4	整数（little-endian）	合约序号。对应YDInstrument的InstrumentRef。
20	1	整数	买卖方向(0：买，1：卖)
21	1	整数	开平标志(0：开仓，1：平仓，3：平今，4：平昨)
22	1	整数	投保标志(1：投机，2：套利，3：套保)

地址偏移	长度(字节)	字段类型	注释
23	1	整数	席位选择方法 (0：YD_CS_Any, 1：YD_CS_Fixed, 2：YD_CS_Prefered)
24	8	IEEE 754双精度浮点类型（little-endian）	报单价格
32	4	整数（little-endian）	报单数量
36	4	整数（little-endian）	投资者报单编号
40	1	整数	报单类型(0：限价单，1：FAK， 2:市价单，3:FOK)
41	1	整数	报单标志
42	1	整数	指定的席位连接编号
43	1	整数	实际使用的席位连接编号
44	4	整数（little-endian）	错误码
48	4	整数（little-endian）	交易所序号
52	4	整数（little-endian）	交易所报单编号
56	4	整数（little-endian）	报单状态
60	4	整数（little-endian）	成交数量
64	4	整数（little-endian）	交易所报单时间
68	4	整数（little-endian）	柜台本地报单编号
72	1	无符号整数	报单组ID
73	1	整数	报单组OrderRef控制方式，详情参见报单组 0：OrderRef单调递增，前后两单OrderRef的间隔必须大于等于1 1：OrderRef严格单调递增，前后两单OrderRef的间隔必须为1
74	1	整数	触发单类型 0：无触发条件 1：止盈触发 2：止损触发
75	13		非公开字段

地址偏移	长度(字节)	字段类型	注释
88	8	IEEE 754双精度浮点类型（little-endian）	触发单触发价格
96	4	整数（little-endian）	报单触发状态 0：未触发 1：已触发
100	4	整数（little-endian）	从该交易日开始（17点整）到报单时间的毫秒数。请参考以下例子： 夜盘21点过500毫秒：3600*(21-17)*1000+500=14400500 日盘9点500毫秒：3600*(24+9-17)*1000+500=57600500 周一日盘9点500毫秒：3600*(24+9-17)*1000+500=5760500 易达时间戳时间和标准时间的转换程序参见ydUtil.h中的string2TimeStamp和timeStamp2String
104	8	整数（little-endian）	全精度的交易所报单编号

6.7.2.3. 成交回报报文

地址偏移	长度(字节)	字段类型	注释
0	16		报文头
16	4	整数（little-endian）	合约序号。对应YDInstrument的InstrumentRef。
20	1	整数	买卖方向(0：买，1：卖)
21	1	整数	开平标志(0：开仓，1：平仓，3：平今，4：平昨)
22	1	整数	投保标志(1：投机，2：套利，3：套保)
23	1		非公开字段
24	4	整数（little-endian）	交易所成交编号
28	4	整数（little-endian）	交易所报单编号

地址偏移	长度(字节)	字段类型	注释
32	8	IEEE 754双精度浮点类型（little-endian）	成交价格
40	4	整数（little-endian）	成交数量
44	4	整数（little-endian）	交易所成交时间
48	8	IEEE 754双精度浮点类型（little-endian）	成交手续费
56	4	整数（little-endian）	柜台本地报单编号
60	4	整数（little-endian）	投资者报单编号
64	1	无符号整数	报单组ID
65	1	整数	实际使用的席位连接编号
66	2		非公开字段
68	4	整数（little-endian）	从该交易日开始（17点整）到成交时间的毫秒数。请参考以下例子： 夜盘21点过500毫秒：3600*(21-17)*1000+500=14400500 日盘9点500毫秒：3600*(24+9-17)*1000+500=57600500 周一日盘9点500毫秒：3600*(24+9-17)*1000+500=5760500 易达时间戳时间和标准时间的转换程序参见ydUtil.h中的string2TimeStamp和timeStamp2String
72	8	整数（little-endian）	全精度的交易所报单编号
80	8	整数（little-endian）	全精度的交易所成交编号

6.7.2.4. 询价回报报文

地址偏移	长度(字节)	字段类型	注释
0	16		报文头
16	4	整数（little-endian）	合约序号。对应YDInstrument的InstrumentRef。

地址偏移	长度(字节)	字段类型	注释
20	4	整数（little-endian）	从该交易日开始（17点整）到询价时间的秒数。请参考以下例子： 夜盘21点整：3600*(21-17)=14400 日盘9点整：3600*(24+9-17)=57600 周一日盘9点整：3600*(24+9-17)=5760 易达整数时间和标准时间的转换程序参见ydUtil.h中的string2TimeID和timeID2String
24	4	整数（little-endian）	询价ID
28	4		非公开字段
32	8	整数（little-endian）	全精度的询价编号

6.7.2.5. 报价回报报文

地址偏移	长度(字节)	字段类型	注释
0	16		报文头
20	4	整数（little-endian）	合约序号。对应YDInstrument的InstrumentRef。
24	1	整数	买开平标志
25	1	整数	买投保标志
26	1	整数	卖开平标志
27	1	整数	卖投保标志
28	8	IEEE 754双精度浮点类型（little-endian）	买价格
36	8	IEEE 754双精度浮点类型（little-endian）	卖价格

地址偏移	长度(字节)	字段类型	注释
44	4	整数（little-endian）	买量
48	4	整数（little-endian）	卖量
52	4	整数（little-endian）	报单客户序号，即OrderRef
56	1	整数	席位选择方法 (0：YD_CS_Any, 1：YD_CS_Fixed, 2：YD_CS_Prefered)
57	1	整数	指定的席位连接编号
58	1	整数	实际使用的席位连接编号
59	1	整数	报价标志 YD_YQF_ResponseOfRFQ：自动填写询价号表示应价，支持的有上期所、能源所、大商所、广期所、郑商所，其他交易所无询价号
60	1	无符号整数	报单组ID
61	1	整数	报单组OrderRef控制方式，详情参见报单组 0：OrderRef单调递增，前后两单OrderRef的间隔必须大于等于1 1：OrderRef严格单调递增，前后两单OrderRef的间隔必须为1
62	2		非公开字段
64	4	整数（little-endian）	错误号
68	4	整数（little-endian）	交易所序号。对应YDExchange的ExchangeRef。
72	4	整数（little-endian）	当ErrorNo为YD_ERROR_InvalidGroupOrderRef时，表示当前柜台已收到的最大OrderRef；否则表示报价编号，用于撤报价。若交易所返回值超长，则会被截断
76	4	整数（little-endian）	买报价的OrderSysID，卖单边报价时为0。若交易所返回值超长，则会被截断
80	4	整数（little-endian）	卖报价的OrderSysID，买单边报价时为0。若交易所返回值超长，则会被截断

地址偏移	长度(字节)	字段类型	注释
84	4	整数（little-endian）	当YDQuoteFlag为YD_YQF_ResponseOfRFQ时，记录了报单响应的询价号，否则为0。若交易所返回值超长，则会被截断
88	4		非公开字段
92	8	整数（little-endian）	全精度的报价编号，用于撤报价
100	8	整数（little-endian）	全精度的买报价的OrderSysID，卖单边报价时为0
108	8	整数（little-endian）	全精度的卖报价的OrderSysID，买单边报价时为0
116	8	整数（little-endian）	当YDQuoteFlag为YD_YQF_ResponseOfRFQ时，记录了全精度的报单响应的询价号，否则为0

6.7.2.6. 撤单或撤报价失败回报报文

由于大量业务复用了撤单或撤报价失败回报报文，因此若报文为撤单失败回报时，请只关注报单标志为0的撤单失败回报，非0的撤单失败回报应忽略。

地址偏移	长度(字节)	字段类型	注释
0	16		报文头
16	4	整数（little-endian）	交易所报单编号或报价编号
20	1	整数（little-endian）	交易所序号。对应YDExchange的ExchangeRef。
21	1	无符号整数	报单组ID
22	1		非公开字段
23	1	整数	报单标志YDOrderFlag，撤报价失败回报报文无效
24	4	整数（little-endian）	错误码

地址偏移	长度(字节)	字段类型	注释
28	4	整数（little-endian)	报价标志IsQuote。 1：本报文为撤报价失败回报报文 0：本报文为撤单失败回报报文
32	4	整数（little-endian)	投资者报单编号OrderRef
36	4		非公开字段
40	8	整数（little-endian)	全精度的交易所报单编号或报价编号

6.7.2.7. 心跳报文

地址偏移	长度(字节)	字段类型	注释
0	16		报文头

6.8. 指定席位

前置是指交易所交易系统中供柜台连接并接收柜台报单和返回回报的服务器，通常会有多个，考虑到每个前置上承载的席位连接数量的差异，以及瞬时业务量的不平衡性，不同前置向交易核心传递报单的性能也会有差异，因此，在交易过程中，投资者应该评估各个前置的性能表现并选择最快的前置报单。

席位是配置在柜台中的用于连接前置的账户，易达会在合规的前提下努力使得每个席位连接不同的前置，以给投资者最大的选择空间。经纪商在部署易达柜台时，通常会配备与交易所前置数相同的席位，因此，通常情况下易达柜台的席位可以覆盖交易所的所有前置，因此，选择前置与选择席位本质上是相同的。

易达柜台中，投资者可以指定上行指令使用的席位，接下来我们将介绍如何获得前置信息，并如何指定席位进行报单。

6.8.1. 获取席位信息

易达柜台支持连接到多个交易所同时交易，每个交易所的席位信息是完全独立的，因此，以下内容均是针对单个交易所而言。

交易所的总席位数可以在YDExchange.ConnectionCount中找到，第一个席位的编号是0，最后一个席位的编号是ConnectionCount-1。易达将席位分为公共席位和专属席位两种：

- 公共席位：可以被所有投资者使用的席位。在YDExchange.IsPublicConnectionID[64]列出了所有公共席位，可以通过席位编号查找该席位是否是公共席位，若IsPublicConnectionID[i]是true则说明第i个席位是公共席位，否则就是专属席位。

- 专属席位：只能被指定的投资者使用，每个专属席位可以被指定给多个投资者，每个投资者也可以拥有多个专属席位。在YDAccountExchangeInfo.IsDedicatedConnectionID[64]列出了每个投资者的专属席位信息，若IsDedicatedConnectionID[i]是true则说明第i个席位是该投资者的专属席位，否则就不是。

综上，投资者的**可选席位**是公共席位和专属席位的叠加，**不可选席位**是属于其他投资者的专属席位。

投资者会有同前置比较两个柜台性能的需求，在API层面没有提供获取席位编号与前置IP地址的方法，如需获取当日对应关系的，请联系经纪商帮助获取该信息。

6.8.2. 指定席位报单

投资者可以设置YDInputOrder和YDInputQuote中的ConnectionSelectionType和ConnectionID以不同的方式选择报单席位。各种类型的席位优选方式的选择逻辑如下所示，请注意，断线、达到流控和不可选的席位不会参与席位选择。

- ConnectionSelectionType为YD_CS_Any时，会实现全局轮询的效果，具体规则为：
    - 会从上一次使用YD_CS_Any报单的席位的下一个席位开始逐一遍历所有席位（遍历遇到最后一个席位后会从头开始，直到遇到遍历的起始席位为止），若找到某个席位没有排队，则直接选中该席位，否则，判断该席位的报单队列长度是否小于之前的所有席位，若是最小的则记录该席位，直到遍历结束，选中记录的队列长度的最小的席位。最后把报单插入选中席位队列中排队等待报送。
    - 轮询并不区分投资者，所有投资者的报单是统一参与轮询的，因此，某投资者使用本模式报单可能并不能从RealConnectionID中获得连续的席位号。
- ConnectionSelectionType为YD_CS_Fixed时，会实现指定席位报单的效果，具体规则为：
    - 直接选中该席位并将报单插入队列排队等待报送
- ConnectionSelectionType为YD_CS_Prefered时，会实现优选选择指定席位报单，遇忙使用其他席位的效果，具体规则为：
    - 会从ConnectionID指定席位开始逐一遍历所有席位（遍历遇到最后一个席位后会从头开始，直到遇到遍历的起始席位为止），若找到某个席位没有排队，则直接选中该席位，否则，判断该席位的报单队列长度是否小于之前的所有席位，若是最小的则记录该席位，直到遍历结束，选中记录的队列长度的最小的席位。最后把报单插入选中席位队列中排队等待报送。

在某些情况下，可能会导致没有席位被选中的情况，易达以不同的错误代码区分了在寻找选中席位时可能存在的多种错误情况：

- 只要有一个席位遇到流控，并且其他席位都处于非专属席位、断线或者到达席位队列上限状态的情况下，则报“YD_ERROR_CanNotSendToExchangeForFlowControl=79没有可用席位，且有席位被流控”的错误
- 如果所有席位都处于非专属席位、断线或者到达队列上限状态的情况下，则报“YD_ERROR_CanNotSendToExchange=9没有可用席位”的错误

当已经在席位队列中排队的报单移动到席位队列的队首，并开始向交易所发送时，如果报单、报价、组合、行权等交易所API因为在途单超过上限或者席位断线返回错误的，则报“YD_ERROR_ExchangeConnectionSendError=80交易所API发送错误”。当发生上述错误且投资者的YDAccountFlag打开了柜台回报YD_AF_NotifyOrderAccept的功能时，收到的notifyOrder回报会随机遇到以下两种情况中的一种，所以投资者的策略程序不应该依赖于回报的个数，而是应该关注回报的状态：

- 收到一个OrderStatus=YD_OS_Rejected和ErrorNo=YD_ERROR_ExchangeConnectionSendError的回报
- 收到两个回报，其中第一个是OrderStatus=YD_OS_Accepted和ErrorNo=0的回报，第二个是OrderStatus=Rejected和ErrorNo=YD_ERROR_ExchangeConnectionSendError的回报

被选中的席位可以通过回报中的YDOrder或者YDQuote中的RealConnectionID获得。

6.8.3. 指定席位撤单

投资者可以设置YDCancelOrder和YDCancelQuote中的ConnectionSelectionType和ConnectionID以不同的方式选择撤单席位，与报单不同，撤单时会受到柜台席位类型的限制而不能完全按照投资者设置进行撤单席位选择。

- 当柜台席位类型为全管理席位时，处理逻辑同报单指定席位的逻辑
- 当柜台席位类型为首席位管理其他席位非管理时，则按序检查原报单席位和首管理席位，若找到某个席位没有排队，则直接选中该席位，否则，选中队列长度较小的席位，若队列长度相同则选中原报单席位。最后把报单插入选中席位队列中排队等待报送。
- 当柜台席位类型为全非管理席位时，撤单必须从原报单席位发出，此时柜台直接选中原报单席位并将报单插入席位队列排队等待报送

与指定席位报单一样，指定席位撤单会以完全相同的原因导致错误，详情请参见指定席位报单。

API中没有渠道能获取在撤单中被选中的席位，如排查问题时需要获取该信息的，请联系经纪商在柜台的inputFlow.txt中帮您查询。

6.8.4. 席位流控

目前各个交易所存在两种限制席位发单速度的席位流控模式，请注意席位流控是不区分投资者的，柜台上所有投资者在同一席位上的报单是汇总计算的：

- 每秒单量流控：限制每个席位或者网关每秒的报撤单笔数上限，若超过限制条件，各交易所API的具体行为是不同，有的交易所API会直接拒单，有的则会缓存报单等到下一秒发送。目前所有交易所都有此席位流控模式，且都公布了流控阈值，具体阈值信息请参见下表
- 在途单流控：限制了已经发送到交易所但是还没有收到回报的报单数量，若超过限制条件，API会直接拒单。目前只有部分交易所有此席位流控模式，且并没有公布流控阈值

交易所	每秒单量流控阈值
中金所、上交所、深交所	50

交易所	每秒单量流控阈值
上期所、能源所、大商所、广期所、郑商所	100

为了避免报单因为每秒单量流控而导致报单被缓存和延误，以及满足尽可能提前报告错误的原则，我们在柜台实现了对每秒单量流控的限制，在报单进入API之前拦截会导致API流控的报撤单，具体的报错信息请参考指定席位报单。但是由于各大交易所没有公布在途单流控的控制方式和阈值，为了确保合规，易达没有实现对在途单流控的限制，而是直接采用交易所API的流控行为。

6.8.5. 席位优选

为了排除最慢的前置，投资者需要比较各个前置的性能，具体评估方法需要由投资者根据策略的具体需求自行实现。单席位选优方法的大致步骤可以是以极小间隔向所有席位发YD_CS_Fixed单，各个席位回报中交易所报单编号OrderSysID最小的是本轮测试最优的席位，通过重复上述步骤并统计分析，选出投资者认可的最佳席位，后续可以使用YD_CS_Fixed和YD_CS_Prefered指定该席位报单。请注意，评估不要过于频繁，否则可能会导致柜台堵单，扰乱本柜台上其他投资者的正常交易，甚至会被交易所认定为异常交易行为。

如果经纪商具备优选能力并希望其客户能使用其优选结果，或者独占柜台中使用其中一个账户进行优选并希望其他账户也能按照优选结果进行报单的，可以使用优选席位上报接口selectConnections定期向柜台传输席位优选结果，分享给同一柜台上的其他账户使用。只有YDAccount.AccountFlag设置了YD_AF_SelectConnection标志的投资者以及管理员才能调用该接口。

```
virtual bool selectConnections(const YDExchange *pExchange,unsigned long long connectionList)=0;
```

connectionList可以看做是一个元素长度为4位序列，每4位表示一个席位编号，最低的位表示最快的席位，必须指定交换的所有席位编号。例如，要设置由快到慢的席位（前置）的顺序为2-3-0-1-4，那么connectionList的二进制位表示为：0100 0001 0000 0011 0010。

优选席位上报的调用间隔不可以超过MaxSelectConnectionGap（默认为5分钟），否则上报结果失效；同时，调用间隔也不能过于频繁，最小间隔不可以低于MinSelectConnectionGap（默认60秒）。MaxSelectConnectionGap和MinSelectConnectionGap的获取方式请参见系统参数。

其他账户想要使用上报结果的，必须使用YD_CS_Any报撤单，在上报结果有效的情况下，席位选择规则为：按照上报结果指定的顺序逐一遍历所有席位，若找到某个席位没有排队，则直接选中该席位，否则，判断该席位的报单队列长度是否小于之前的所有席位，若是最小的则记录该席位，直到遍历结束，选中记录的队列长度的最小的席位。最后把报单插入选中席位队列中排队等待报送。

6.9. 紧急平仓

当市场处于极端情况时，会遇到开盘无法平仓或者平仓困难的情况，为此易达提供了平仓缓冻结的紧急平仓功能。该功能可以使得投资者或者经纪商在开盘时以更高的频率报送平仓报单，从而使得平仓报单能排在前面而更容易成功平仓。投资者的YDAccount.AccountFlag必须在设置了YD_AF_NoCloseFrozenOnInsertOrder以后才能使用。

投资者在启用该功能后，柜台在收到该投资者的平仓指令后不会冻结持仓，只有在收到交易所报单成功的报单回报后冻结持仓，这样，盘前投资者可以同时发起多个平仓指令，一旦收到报单成功的回报就会立即冻结持仓，阻止后续平仓指令发送。平仓缓冻结只是冻结时间延后，并不代表不检查持仓，如果投资者的报单量超过持仓量，报单是无法通过仓位检查的，报单会报错。举例来说，假设某投资者有某合约持仓2手，在收到回报之前，他可以报送多个1手和2手的平仓报单，但是无法报送任何超过2手的平仓报单。

紧急平仓功能可能会给交易所带来大量错单，请谨慎使用本功能，切勿将本功能用于开盘抢单等正常业务情景，易达对因误用引起的监管处罚不承担任何责任。同时，在全非管理席位的系统中，平仓缓冻结功能会遇到无法解决的问题，请尽可能不要在全非管理席位的系统中使用，错误场景如下：假设某投资者先卖平、买开、卖平，如果最终是两个卖平先到，最终该客户则会导致持仓数据不正确。

6.10. 超时未知单处理

超时未知单是指柜台向交易所发单超过一定时间后没有收到交易所回报的报单，因为没有交易所回报，所以报单的状态不会发生改变，该报单会停留在柜台中持续冻结保证金或者持仓而无法释放。超时未知单通常发生在与交易所断线时候报单或者交易所切换状态时，但是发生概率极低。易达提供了处理超时未知单的功能，但是在使用时请严格按照以下说明进行，在确认是超时未知单的基础上再进行处理，以免造成损失。

易达柜台会持续监测处于YD_OS_Accepted的报单（询价单和大商所、广期所YDOrderFlag为YD_YOF_Mark的指令除外），若该报单已经超过MissingOrderGap（默认值为60秒，经纪商可以修改，详情参见系统参数）没有收到柜台回报，那么就会通过notifyMissingOrder告知投资者有一个指超时未知单：

```
1 virtual void notifyMissingOrder(const YDMissingOrder *pMissingOrder)
```

超时未知单结构YDMissingOrder与YDOrder是相同的，但是在返回时YDMissingOrder.InsertTime填写的是柜台收到的时间，这点与YDOrder.InsertTime是交易所时间是有区别的，请注意区分。

投资者在收到通知后，请先在主席柜台确认该报单的状态：

- 如果主席上有该报单的信息，那么很显然该报单已经被交易所接收且交易所给出的回报已经丢失
- 如果主席上没有该报单的信息，则说明要么该回报没有发送到交易所，要么交易所还没有发送回报，考虑到生产中交易所曾经发生过回报超过1分钟还没有回来的情况，所以建议再等待一段时间

如果已经确认该报单是超时未知单，那么请联系经纪商使其帮助处理，经纪商业务人员也应该遵守上述规则经过**审慎**判断后才能帮助投资者撤销该超时未知单。超时未知单只能由经纪商的管理员账户撤销，投资者自身不可撤销。在经纪商执行了超时未知单撤单操作后，该超时未知单的撤单回报会通过notifyOrder返回，返回的YDOrder.ErrorNo为YD_ERROR_InternalRejected，YDOrder.OrderStatus为YD_OS_Rejected。

如果在撤回了“超时未知单”后收到了交易所的报单回报，易达柜台会将其当做外部报单处理，而外部报单的OrderRef和OrderLocalID都为-1，这样就失去了和原报单关联的机会，当然此时投资者是可以撤回该报单的。如果后续继续收到了成交回报，易达会更新持仓并向投资者发送成交回报，因此投资者不用担心持仓和资金的正确性。

6.11. 性能调优

性能调优的目标是降低总体穿透，即从接收到行情开始，到报单请求从柜台发出到交易所，包含了行情接收、策略计算、客户端发单、交换机转发、柜台风控并发送到交易所等步骤。上述的每个步骤都在交易的关键路径上，一个步骤的低性能将会抵消其他步骤在高性能方面的努力，只有每个步骤都做到最高穿透性能，才能保证领先于市场。

其中，客户端发单和柜台风控并发送到交易所是易达提供的交易服务的关键步骤，这两个步骤的穿透时间我们分别称为API穿透性能和柜台穿透性能。

6.11.1. API穿透性能

API穿透性能是指开始调用insertOrder(或者是其他报单方法)到报单的第一个字节出现在光纤上的时间差。

由于大部分柜台的API在调用后即刻返回，想要测试API穿透性能的唯一办法是将投资者策略机上行柜台的TX口分光后回接到同一个柜台上，在报单前记录下系统时钟的时间戳，然后在保持网卡时钟与系统时钟的同步的状态下给从分光口上收到的每个报单包打上时间戳，两个时间戳相减就是API的穿透性能的时间戳。

我们当然可以按照上述方法来测试易达API的穿透性能，但是由于易达API的发单函数返回的时候就已经将包发送到了光纤上，因此可以直接在报单前后记录系统时间戳然后相减的方式来取得穿透性能。投资者可以用两种方法同时测试以验证简易测试方法的正确性和准确性。

易达提供了快速且高精度的时间戳函数，性能开销小，不会对报单关键路径产生过多的影响，在测试API穿透性能时，通过在报单函数前后使用该函数获得时间戳，相减后即得到了API穿透的纳秒数。

```
// Returns nanoseconds elapsed since current process starts up
YD_API_EXPORT unsigned long long getYDNanoTimestamp();
```

易达API经过长时间的迭代优化，其在使用Solarflare X2522或者Exanic X10/X25网卡并配合其加速软件的情况下，API穿透性能可以达到250ns以下，若投资者实际测得的穿透性能没有达到这个指标的，请首先检查网卡、启动方式是否符合易达要求并努力达到易达官方的性能标准，如仍无法解决的请通过经纪商与我们联系。

曾经在其他系统上使用过裸协议的投资者会倾向于使用裸协议，易达为了照顾投资者的交易习惯也提供了裸协议报撤单的方法，但是我们认为易达的穿透性能已经达到了普通FPGA的水准，因此，如果投资者尝试自己实现裸协议报单，那么请在实现完成裸协议报单后与我们的API的穿透性能进行比较，最终选择较快的一种作为您生产报单的方式。

6.11.2. 柜台穿透性能

柜台穿透性能指报单第一个字节进入和离开柜台的时间差，该指标可客观公正的衡量柜台的性能，是最常用的判断柜台性能的指标。

业内通常使用分光器或者端口镜像的方式进行测量。分光器精度高成本低但不易调整，适合于临时搭建测量；端口镜像易于调整，但只有Arista 7130等高端型号提供的端口镜像镜像功能可同时做到对最小性能影响和最高测量精度，其他中低端型号并不适合用来测量柜台穿透。易达向经纪商提供了专门的工具用来测量和分析柜台上每笔报单、报价和撤单的上行穿透性能，如需了解您的报单穿透数据可向经纪商索取。

绝大多数情况下，穿透性能不应该是投资者应该关注的问题，易达对每个版本都做了全方位的测试，确保在生产环境中可以达到易达预期的性能指标。当投资者怀疑因穿透性能恶化而导致收益下降时，可与经纪商联系查看穿透是否符合性能标准。标准性能会随每台服务器测试完成时提供给经纪商。

在某些特殊的交易行为情况下，柜台的穿透性能会恶化。目前已知柜台上的报单发送速度太快或者太慢都有可能会显著增加穿透延时，请分别予以应对。若不是这些交易行为引起的穿透性能恶化，请及时联系易达客服排查解决。

6.11.2.1. 堵单

堵单发生于多个报单以低于穿透延时的间隔到达柜台，且指定从同一个席位发出。由于交易所是TCP连接，报单需要排队发出，一方面同时到达，一方面排队发出，这就导致了从穿透性能上会体现为，除了第一张单子穿透延时正常外，接下来每张报单的穿透延都会相对上一张报单增加一个较为固定的延时，这样就形成了堵单。

发生堵单可能是以下情况的一种或者多种情况的叠加。

- 报单都选择同一个席位用Fix排队发出（Any和Preferred则不会排队）。特别容易出现在做了席位优选的条件下，不同的投资者会不约而同持续使用其优选的最佳席位报单。
- 有投资者过于频繁的报单，导致席位处于忙的总体时间增加，造成后续Fix指令有更大的几率排队。例如有客户在以较高频率发送席位优选单且没有避开行情切片的到达时间。
- 行情切片到来或者交易节开始起到了发令枪的效果，大部分投资者都会在同一时间点抢发单。
- 有投资者使用了错误的预热方法导致柜台收到巨大的报单量。例如，以极高的频率发送撤单到柜台以期望获取预热效果。
- 同一时间向不同的席位发单也会导致延时的依次增加，但增加量相对同一席位的情况缓解很多，比较常见的情况是在对所有席位发优选单时，后面席位的优选单穿透会少许增加。

堵单本质上是正常的交易行为引起的，易达除了进一步降低穿透外，没有更好的办法来缓解这类问题。对于经纪商，如果发现有投资者采取的错误的预热方式可以及时劝阻，有投资者席位优选单频率过高或者没有避开行情时建议他调整优选单的发送时机，将互相冲突的严重的投资者分开在不同的柜台上。

6.11.2.2. 慢速报单

柜台上的报单速度太慢时会导致柜台服务器的缓存变冷，新的报单来的时候该报单的穿透会显著增加。绝大多数时候柜台不会遇到这个问题，因为在柜台上交易的所有投资者的报单都会使缓存变热，只有柜台上所有投资者都以极低的速度（例如整个柜台1单/分钟的频率）报单时，才可能出现这个问题。当投资者与经纪商确认确实遇到这种情况时，我们建议投资者提前发送预热单以规避这个问题。下面我们将说明预热单的发送方式和发送时机。

预热单和正式单走相同的执行路径能获得最佳预热效果。

- 预热单和正式单类型相同时执行路径相同，预热效果最好。例如，正式单是报单时，用报单预热的效果优于撤单的。
- 预热单的预热效果仅限于预热单指定的席位，对其他席位的预热效果有限。
- 没能到达交易所的预热单的执行路径短于正式单，预热效果不能达到最佳。
- 预热单和正式单从同一个席位出去但是使用的不同的合约，此时执行路径是相同的，可以达到预热效果。

预热单和正式报单的间隔也会影响预热效果，测试表明两者间隔5秒发送就可以获得最佳的预热效果，更短的间隔并不能获得更好的效果，反而会在收到回报前冻结更长时间的报单保证金，也会增加柜台的压力甚至堵单。同时，预热单要避开行情，否则可能堵住正常的单子，由于预热单并不密集，所以做到这一点相对容易。

不同投资者的预热效果互不影响，不用担心其他客户的预热行为会影响到您的预热效果，反而对您的正式报单也有一定的预热效果。

根据上述预热效果的发送条件，即预热单要做到和正式单类型和席位相同，计算出行情到来的时间提前发送预热单，确保其发送到交易所，这种精确预热是我们最推荐的做法。对于在定时发送席位优选单的投资者，席位优选单也是一种可行的预热单，但效果不如精确预热。

7. 行情

易达柜台从交易所前置获取一档行情并转发给投资者，易达提供行情主要用于持仓盈亏、保证金的刷新计算，以及在组播行情中断的情况下提供备用行情使用，其相比交易所组播行情较慢，可能会错过交易机会，不建议直接使用易达转发的行情进行交易。

对于RecalcMode设置为auto或者subscribeOnly的投资者，获取**持仓合约行情**最简单的方法是通过合约指针中的行情指针m_pMarketData直接查询对应合约的行情，详情请参见资金刷新机制章节内容。

对于需要获取非持仓合约行情，或者RecalcMode为off，或者需要在行情更新时收到通知的投资者，可以通过自行主动订阅的方式实现。主动订阅行情前，需要确保API参数按照如下方式配置（RecalcMode设置为auto或者subscribeOnly时API会覆盖以下参数，所以RecalcMode设置为auto或者subscribeOnly时可以不用关心API行情参数的设置）。因为易达柜台没有开放UDP行情，因此ReceiveUDPMarketData必须始终保持为no，否则将收不到任何行情。

```
ConnectTCPMarketData=yes
ReceiveUDPMarketData=no
```

当ConnectTCPMarketData设置为yes时，启动API后会创建一个单独的行情线程，为了避免行情线程影响到策略系统中其他线程的运行，可以配置API参数将行情线程绑定到某个CPU上面。

```
# Affinity CPU ID for thread to receive TCP market data, -1 indicate no need to set CPU affinity
TCPMarketDataCPUID=-1
```

如果希望行情尽可能快得通知到策略程序，可以设置行情接收线程的工作方式：

- 当设置为-1时系统处于忙查询状态，行情通知性能最佳，但是线程将占满运行所在CPU核心
- 如果不希望行情线程占用过多CPU，且对于行情通知性能并不特别在意的投资者，可以设置select函数的超时时间，这种模式相比忙查询较慢，但是CPU占用将大大降低。我们推荐在GUI客户端中设置超时时间，以减少不必要的性能开销。

```
# Timeout of select() when receiving TCP market data, in millisec. -1 indicates running without select()
TCPMarketDataTimeout=10
```

关于上述参数的详细说明可以参见行情配置。

以下API提供了订阅和退订某一合约行情的功能，凡是主动订阅了的合约行情，都可以通过上述合约指针中的行情指针m_pMarketData直接查询。

```
virtual bool subscribe(const YDInstrument *pInstrument)=0;
virtual bool unsubscribe(const YDInstrument *pInstrument)=0;
```

合约行情更新后，会通过以下回调函数通知到投资者。

```
virtual void notifyMarketData(const YDMarketData *pMarketData) {}
```

8. 风控

易达的风控规则分为事前风控和事后风控两类。

事前风控会拦截不合规的报单，使其不能到达交易所，事前风控通常是监管要求的风控，一旦违反可能会引起禁止交易等监管处罚，如自成交、撤单数限制、仓位限制等。事前风控由柜台在向交易所发送指令前执行，若不通过将会向API返回错单回报。使用ydExtendedApi的投资者可以使用checkAndInsert系列方法在API本地执行风控检查，若通过则正常向柜台发单，若失败则函数返回False，投资者可以在YDInputOrder或者YDInputQuote内查看ErrorNo。需要注意的是，即便API本地执行了风控检查，柜台在处理报单的过程中仍然会再次执行风控检查。

事后风控是经纪商或者投资者自身关注的风控，违反这些风控规则会增加经纪商和投资者的风险，因为对风控阈值的边界是较为宽松的，即便稍微突破一点也不会有质的变化，如信息量等风控规则，一般的处置方式是自动调整投资者的交易权限或者发送风控报警。事后风控规则由管理端中的事后风控模块执行。

8.1. 事前风控

8.1.1. 开仓量风控

开仓量风控分为产品层和合约层两个维度，每个维度上可以分别对开仓量、买开仓量和卖开仓量进行控制。下面以开仓量为例进行说明，买开仓量和卖开仓量风控的处理方法是类似的。

产品层的风控规则是对属于该产品的所有合约的开仓量加总进行汇总风控，一旦触发了产品层风控阈值，该产品内所有合约都无法继续发送开仓指令。

合约层的风控规则是针对单个合约的开仓量加总后进行风控，一旦触发了合约层风控阈值，该合约无法继续发送开仓指令。

本风控并不是在开仓量到达阈值以后才开始风控，否则新的开仓报单一旦报到交易所就会超过风控阈值，因此，我们实际上控制的是可能开仓量，当开仓报单报出去以后我们就会增加可能开仓量，处于挂单状态时不改变可能开仓量，报单状态达到终态后从可能开仓量中扣减未成交开仓量。因此，如果当前有大量开仓挂单也会导致后续开仓报单因为本风控而被拒绝。

产品层的各个风控参数可以参考YDAccountProductInfo.TradingConstraints[HedgeFlag]的OpenLimit以及DirectionOpenLimit[2]，合约层的各个风控参数可以参考YDAccountInstrumentInfo.TradingConstraints[HedgeFlag]的OpenLimit以及DirectionOpenLimit[2]，具体字段含义请参考风控参数。

8.1.2. 成交量风控

成交量风控分为产品层和合约层两个维度。

产品层的风控规则是对属于该产品的所有合约的买卖开平成交量加总进行汇总风控，一旦触发了产品层风控阈值，该产品内所有合约都无法继续发送任何指令。

合约层的风控规则是针对单个合约的买卖开平成交量加总后进行风控，一旦触发了合约层风控阈值，该合约无法继续发送任何指令。

本风控并不是在成交量到达阈值以后才开始风控，否则新的报单一旦报到交易所就会超过风控阈值，因此，我们实际上控制的是可能成交量，当报单报出去以后我们就会增加可能成交量，处于挂单状态不改变可能成交量，报单状态达到终态后从可能成交量中扣减未成交报单量。因此，如果当前有大量挂单也会导致后续报单因为本风控而被拒绝。

产品层的各个风控参数可以参考YDAccountProductInfo.TradingConstraints[HedgeFlag]的TradeVolumeLimit，合约层的各个风控参数可以参考YDAccountInstrumentInfo.TradingConstraints[HedgeFlag]的TradeVolumeLimit，具体字段含义请参考风控参数。

8.1.3. 撤单笔数风控

撤单笔数风控分为产品层和合约层两个维度。

产品层的风控规则是对属于该产品的所有合约的限价报单的撤单笔数加总进行汇总风控，一旦触发了产品层风控阈值，该产品内所有合约都无法继续发送任何指令。

合约层的风控规则是针对单个合约的限价报单的撤单笔数加总后进行风控，一旦触发了合约层风控阈值，该合约无法继续发送任何指令。

本风控并不是在限价报单的撤单笔数到达阈值以后才开始风控，否则新的报单一旦报到交易所就会超过风控阈值，因此，我们实际上控制的是可能撤单笔数，当限价报单报出去以后我们就会增加可能撤单笔数，处于挂单状态时或者主动撤单则不改变可能撤单笔数，若全部成交则将撤单笔数扣减一笔。因此，如果当前有大量限价挂单也会导致后续限价报单因为本风控而被拒绝。

产品层的各个风控参数可以参考YDAccountProductInfo.TradingConstraints[HedgeFlag]的CancelLimit，合约层的各个风控参数可以参考YDAccountInstrumentInfo.TradingConstraints[HedgeFlag]的CancelLimit，具体字段含义请参考风控参数。

8.1.4. 持仓量风控

持仓量风控分为产品层和合约层两个维度，每个维度上可以分别对持仓量、买持仓量和卖持仓量进行控制。下面以持仓量为例进行说明，买持仓量和卖持仓量风控的处理方法是类似的。

产品层的风控规则是对属于该产品的所有合约的买卖持仓量加总进行汇总风控，一旦触发了产品层风控阈值，该产品内所有合约都无法继续发送开仓指令。

合约层的风控规则是针对单个合约的买卖持仓量加总后进行风控，一旦触发了合约层风控阈值，该合约无法继续发送开仓指令。

本风控并不是在持仓量到达阈值以后才开始风控，否则新的开仓报单一旦报到交易所就会超过风控阈值，因此，我们实际上控制的是可能持仓量，当开仓报单报出去以后我们就会增加可能持仓量，处于挂单状态时不改变可能持仓量，报单状态达到终态后从可能持仓量中扣减未成交报单量。因此，如果当前有大量开仓挂单也会导致后续开仓报单因为本风控而被拒绝。

产品层的各个风控参数可以参考YDAccountProductInfo.TradingConstraints[HedgeFlag]的PositionLimit以及DirectionPositionLimit[2]，合约层的各个风控参数可以参考YDAccountInstrumentInfo.TradingConstraints[HedgeFlag]的PositionLimit以及DirectionPositionLimit[2]，具体字段含义请参考风控参数。

8.1.5. 单笔报单量风控

虽然交易所控制了每个合约的最大报单量，但是对于某些投资者的某些策略还是偏大，需要由柜台协助控制，为此易达支持限制每笔报单的最大下单量。

报单量风控的参数在通用风控参数中，分别支持在交易所、产品以及合约三个维度上进行设置：

层级	字段	说明
交易所级	GeneralRiskParamType	固定为YD_GRPT_ExchangeMaxOrderVolume
	AccountRef	-1表示对所有投资者生效，否则表示用户的AccountRef值，对于投资者账户登录来说就是该账户本身
	ExtendedRef	对应于YDExchange.ExchangeRef，唯一确定交易所
	IntValue1	报单量上限
产品级	GeneralRiskParamType	固定为YD_GRPT_ProductMaxOrderVolume
	AccountRef	-1表示对所有投资者生效，否则表示用户的AccountRef值，对于投资者账户登录来说就是该账户本身
	ExtendedRef	对应于YDProduct.ProductRef，唯一确定产品
	IntValue1	报单量上限
合约级	GeneralRiskParamType	固定为YD_GRPT_InstrumentMaxOrderVolume
	AccountRef	-1表示对所有投资者生效，否则表示用户的AccountRef值，对于投资者账户登录来说就是该账户本身
	ExtendedRef	对应于YDInstrument.InstrumentRef，唯一确定合约
	IntValue1	报单量上限

由于各个层级中AccountRef可以为全体账户或者指定账户两种，因此两两组合一共六个层级，他们优先级从高到低（高优先级覆盖低优先级的参数配置）分别是：

- 合约级且指定账户
- 产品级且指定账户
- 交易所级且指定账户

- 合约所级且对所有投资者
- 产品所级且对所有投资者
- 交易所级且对所有投资者

8.1.6. 价格偏离度风控

当报单价格与动态参考价之差超过阈值，或者当报单价格与最新价之差超过阈值，该报单会被柜台拒绝。其中，动态参考价在不同交易所的取值规则不同：

- 对于上交所和深交所，动态参考价是指合约在最近一次集合竞价阶段产生的成交价格，开盘集合竞价阶段未产生成交价格的，以前结算价格作为最近参考价格；盘中集合竞价阶段未产生成交价格的，以进入该集合竞价阶段前的最后一笔成交价格作为最近参考价格
- 对于期货交易所，若行情中的成交量不为0时，动态参考价取最新价，否则取昨结算价

易达在行情中直接提供了动态参考价YDMarketData.DynamicBasePrice。

本风控规则中涉及到大量参数，满足以下任何一条均会触发风控规则：

- $\text{报单价} \geq \text{动态参考价} \times (1 + \text{动态参考价偏离比例上限})$
- $\text{报单价} \leq \text{动态参考价} \times (1 - \text{动态参考价偏离比例下限})$
- $\text{报单价} \geq \text{动态参考价} \times (1 + \text{动态参考价偏离}tick\text{数上限})$
- $\text{报单价} \leq \text{动态参考价} \times (1 - \text{动态参考价偏离}tick\text{数下限})$
- $\text{报单价} \geq \text{最新价} \times (1 + \text{最新价偏离比例上限})$
- $\text{报单价} \leq \text{最新价} \times (1 - \text{最新价偏离比例下限})$
- $\text{报单价} \geq \text{最新价} \times (1 + \text{最新价偏离}tick\text{数上限})$
- $\text{报单价} \leq \text{最新价} \times (1 - \text{最新价偏离}tick\text{数下限})$

由于字段限制，易达需要通过多条配置为相同产品的通用风控规则合成表示一条本风控规则，由于本风控规则只允许配置在产品层面（即ExtendedRef只能表示为一个产品），且不允许指定账户，因此下文中的AccountRef和ExtendedRef均被略去，价格偏离度的其余参数如下表所示：

层级	字段	说明
动态参考价偏离比例上限	GeneralRiskParamType	固定为YD_GRPT_DynamicPriceLimitUpperRatio
	FloatValue	比例阈值
动态参考价偏离比例下限	GeneralRiskParamType	固定为YD_GRPT_DynamicPriceLimitLowerRatio
	FloatValue	比例阈值
动态参考价偏离tick数上限	GeneralRiskParamType	固定为YD_GRPT_DynamicPriceLimitUpperTickCount
	IntValue1	tick数

层级	字段	说明
动态参考价偏离tick数下限	GeneralRiskParamType	固定为YD_GRPT_DynamicPriceLimitLowerTickCount
	IntValue1	tick数
最新价偏离比例上限	GeneralRiskParamType	固定为YD_GRPT_DynamicLastPriceLimitUpperRatio
	FloatValue	比例阈值
最新价偏离比例下限	GeneralRiskParamType	固定为YD_GRPT_DynamicLastPriceLimitLowerRatio
	FloatValue	比例阈值
最新价偏离tick数上限	GeneralRiskParamType	固定为YD_GRPT_DynamicLastPriceLimitUpperTickCount
	IntValue1	tick数
最新价偏离tick数下限	GeneralRiskParamType	固定为YD_GRPT_DynamicLastPriceLimitLowerTickCount
	IntValue1	tick数

8.1.7. 期权买入额度

本风控规则可以控制期权买入额度（期权多头持仓总成本的规模），期权多头的持仓成本是把相应维度上（某账户全部或者某账户在某交易所上）期权多头持仓的持仓成本相加得到的，单条期权多头的持仓成本为其持仓明细的总和，单条持仓明细的成本为其成交价格乘以成交量。

本风控规则可以在某账户全部或者某账户在某交易所上分别设置阈值，上面这两个维度的期权多头持仓成本也是分别计算和控制的，违反任意维度的阈值都将导致拒单。

全局级的风控参数如下表所示：

层级	字段	说明
全局级	GeneralRiskParamType	固定为YD_GRPT_OptionLongPositionCost
	AccountRef	-1表示对所有投资者生效，否则表示用户的AccountRef值，对于投资者账户登录来说就是该账户本身
	FloatValue	持仓成本限额阈值

如果ydExtendedApi，则可以通过YDExtendedAccount.OptionLongPositionCostLimit查到该投资者全局层面的持仓成本阈值，也可以通过YDExtendedAccount.OptionLongPositionCost查到该投资者全局层面的当前持仓成本汇总值。

由于AccountRef可以为全体账户或者指定账户两种，因此一共两个层级，他们优先级从高到低（高优先级覆盖低优先级的参数配置）分别是：

- 指定账户
- 对所有投资者

交易所级的风控参数如下表所示：

层级	字段	说明
交易所级	GeneralRiskParamType	固定为YD_GRPT_ExchangeOptionLongPositionCost
	AccountRef	-1表示对所有投资者生效，否则表示用户的AccountRef值，对于投资者账户登录来说就是该账户本身
	ExtendedRef	对应于YDExchange.ExchangeRef，唯一确定交易所
	FloatValue	持仓成本限额阈值

由于AccountRef可以为全体账户或者指定账户两种，因此一共两个层级，他们优先级从高到低（高优先级覆盖低优先级的参数配置）分别是：

- 指定账户
- 对所有投资者

8.2. 事后风控

8.2.1. 错单笔数风控

交易所对错单笔数是有监管的，但是监管措施较为灵活，为了避免这种情况的发生，经纪商通常希望能控制投资者的交易所错单笔数，而且即便稍微超过一点也问题不大，因此，易达使用事后风控来控制交易所错单笔数，在达到错单笔数阈值后，易达系统会将该客户的交易权限设置为禁止交易。请注意只有从交易所返回的错单才增加错单笔数总数，被柜台拦截的错单不会增加错单笔数。

本风控规则在计算交易所错单时是不精确的，如果发生了ydServer的重启，而客户端没有使用高可用模式恢复，那么前一次ydServer运行时发生的错单将不被计数。

全局级的风控参数如下表所示：

层级	字段	说明
全局级	GeneralRiskParamType	固定为1004
	AccountID	空表示对所有投资者生效，否则表示用户的AccountID值

层级	字段	说明
	FloatValue	错单笔数阈值

由于AccountID可以为全体账户或者指定账户两种，因此一共两个层级，他们优先级从高到低（高优先级覆盖低优先级的参数配置）分别是：

- 指定账户
- 对所有投资者

目前本风控规则由外部风控系统负责执行，因此投资者无法通过API获取具体风控参数值，也无法获知风控规则是否触发。

8.2.2. 成交持仓比风控

当某产品内所有合约的成交量达到一定规模时，则该产品的成交量/持仓量的比值不能超过一定比值，否则会被视为违规，当触发风控规则后，易达会向经纪商的业务人员发出警告以引起其关注，并不会有刚性的风控措施。本风控规则只对上交所和深交所的期权有效。

成交量为某产品的所有合约上买卖开平的成交的成交量的总数。

持仓量为某产品的所有合约的实时净持仓或日初净持仓两者取大，即 $max(某产品的所有合约的日初净持仓总和，某产品的所有合约的实时净持仓总和)$。单个合约净持仓的计算公式为：$|多头持仓量 - 空头持仓量| + 多头冻结量 + 空头冻结量$，其中多头和空头持仓量为扣除相应的冻结持仓量以后的值，冻结量主要包括因为平仓报单、组合等原因导致的仓位冻结，但是在以下两种情况下，不计入冻结量：

- 当天是最后交易日，该合约的冻结量为0
- 距离最后到期日还剩2天时，且当组合类型为认购牛市价差、认购熊市价差、认沽牛市价差、认沽熊市价差的，不计入组合部分的冻结量，因为根据交易规则当天结算时交易所会自动解组这些组合

> 假设日初净持仓为5手，目前A合约有权利仓11手，义务仓5手，B合约有权利仓2手，义务仓3手，此时下一个 认购牛市价差组合，第一腿为A的权利仓，第二腿为B的义务仓，则计算持仓量的过程如下：
>
> 实时净持仓A=|(11-1)-5|+1=5，其中两个括号内的减一是组合冻结的仓位
>
> 实时净持仓B=|2-(3-1)|+1=1，其中两个括号内的减一是组合冻结的仓位
>
> 实时净持仓=5+1=6
>
> 净持仓=max(5, 6)=6

风控参数如下表所示：

层级	字段	说明

层级	字段	说明
产品级	GeneralRiskParamType	固定为YD_GRPT_TradePositionRatio
	AccountRef	-1表示对所有投资者生效，否则表示用户的AccountRef值，对于投资者账户登录来说就是该账户本身
	ExtendedRef	对应于YDProduct.ProductRef，唯一确定产品
	IntValue1	成交量规模阈值，当总成交量达到阈值时才进行成交持仓比的计算
	FloatValue	成交持仓比阈值

由于AccountRef可以为全体账户或者指定账户两种，因此一共两个层级，他们优先级从高到低（高优先级覆盖低优先级的参数配置）分别是：

- 指定账户
- 对所有投资者

目前本风控规则由ydClient负责执行，投资者可以通过ydClient查看风控参数与风险规则状态，也可以通过API获取风控参数。

8.2.3. 撤单报单比风控

当某产品内所有合约的报单笔数之和达到一定规模时（假设为X），则该产品的(撤单笔数-X)/(报单笔数-X)的比值不能超过一定比值，否则会被视为违规，当触发风控规则后，易达会向经纪商的业务人员发出警告以引起其关注，并不会有刚性的风控措施。其中，撤单笔数为属于该产品的所有合约上的撤单笔数之和，报单笔数为属于该产品的所有合约上的报单笔数之和。

风控参数如下表所示：

层级	字段	说明
产品级	GeneralRiskParamType	固定为YD_GRPT_OrderCancelRatio
	AccountRef	-1表示对所有投资者生效，否则表示用户的AccountRef值，对于投资者账户登录来说就是该账户本身
	ExtendedRef	对应于YDProduct.ProductRef，唯一确定产品
	IntValue1	报单笔数阈值，当总报单笔数达到阈值时才进行撤单报单比的计算

层级	字段	说明
	FloatValue	撤单报单比阈值

由于AccountRef可以为全体账户或者指定账户两种，因此一共两个层级，他们优先级从高到低（高优先级覆盖低优先级的参数配置）分别是：

- 指定账户
- 对所有投资者

目前本风控规则由ydClient负责执行，投资者可以通过ydClient查看风控参数与风险规则状态，也可以通过API获取风控参数。

8.2.4. 信息量风控

目前上期所、大商所、郑商所都放弃了硬性的撤单数控制转而使用信息量控制投资者的报撤单数量，以减少无效报单对交易所的压力。不少投资者对此转变并不适应，甚至投资者可能因为没有控制好报单量导致支付了高额的信息量手续费。考虑到主席柜台尚未在盘中收取信息量手续费，而且其全市场汇总计算的业务特性也导致短期内信息量手续费的计算较难实现，因此，易达提供了信息量的事后风控，尽可能帮助经纪商和投资者减少信息量手续费的风险。按照监管规定，信息量业务中的信息量和OTR（报单成交比）的计算方式如下。

各种不同类型报单的信息量如下：

- 限价单：若全部成交仅计1笔报单笔数；若撤单，则计1笔报单笔数和1笔撤单笔数。
- FAK/FOK/市价单：若全部成交仅计1笔报单笔数；若未成交或未全部成交而产生撤单，则计1笔报单笔数和1笔撤单笔数。
- 询价单，上期所、能源所增加1笔信息量，其他交易所不计。投资者可以在AccountInstrumentInfo.RFQCount查看合约的询价单报单量。
- 组合合约：组合合约的各腿合约信息量分别计入各腿合约上
- 交易所强平单，非期货公司强平单：计入信息量
- 强制减仓单：不计入信息量
- 错误报单和限价单的撤单：不计入信息量
- 组合和解组、期权行权和放弃行权、期权对冲、履约后对冲的报单、撤单、错误报单、错误撤单：不计入信息量

为了确保OTR在任何情况下都是有意义的，我们使用如下OTR的计算公式，基本上与监管的公式一致，同时保证了在极端条件下OTR的合理性：$OTR = max(\text{信息量}, 1)/max(\text{有成交的报单数}, 1) - 1$，若某笔报单部分或全部成交，则该笔报单计为1笔有成交的报单，1笔报单若分多次成交不重复统计。

易达提供了两种不同的信息量控制方式，分别是单信息量控制和信息量与OTR共同控制。

单信息量控制时，只要信息量超过设定的阈值就会按照不同的阈值设置投资者在该合约上面的交易权限为只能平仓或者禁止交易。其风控参数如下，这些参数在外部风控程序中设置，目前不会下发到API：

层级	字段	说明

层级	字段	说明
产品级	GeneralRiskParamType	固定为1001
	AccountID	空表示对所有投资者生效，否则表示用户的AccountID值空
	ExtendedID	对应于YDProduct.ProductID，唯一确定产品
	IntValue1	信息量阈值，触发后设置该合约只能平仓权限
	IntValue2	信息量阈值，触发后设置该合约禁止交易权限

由于AccountID可以为全体账户或者指定账户两种，因此一共两个层级，他们优先级从高到低（高优先级覆盖低优先级的参数配置）分别是：

- 指定账户
- 对所有投资者

目前本风控规则由外部风控系统负责执行，因此投资者无法通过API获取具体风控参数值，也无法获知风控规则是否触发。

信息量与OTR共同控制时，只有当OTR和信息量都超过设定的阈值时，才会按照不同的阈值设置投资者在该合约上面的交易权限为只能平仓或者禁止交易。其风控参数如下：

层级	字段	说明
产品级	GeneralRiskParamType	固定为1002
	AccountID	空表示对所有投资者生效，否则表示用户的AccountID值
	ExtendedID	对应于YDProduct.ProductID，唯一确定产品
	IntValue1	信息量阈值，触发后设置该合约只能平仓权限
	IntValue2	信息量阈值，触发后设置该合约禁止交易权限
	FloatValue	OTR阈值

由于AccountID可以为全体账户或者指定账户两种，因此一共两个层级，他们优先级从高到低（高优先级覆盖低优先级的参数配置）分别是：

- 指定账户
- 对所有投资者

目前本风控规则由外部风控系统负责执行，因此投资者无法通过API获取具体风控参数值，也无法获知风控规则是否触发。

8.2.5. 临近到期日禁止开仓

为方便经纪商为临近到期日的合约设置禁止开仓的交易权限，易达借助事后风控规则实现了便捷的设置方法。

本风控规则会根据YDInstrument.ExpireTradingDayCount判断是否到达了设定的阈值，若到达则会相应设置各个投资者上对应合约的只能平仓交易权限。

本风控规则的参数如下：

层级	字段	说明
产品级	GeneralRiskParamType	固定为1003
	AccountID	空表示对所有投资者生效，否则表示用户的AccountID值
	ExtendedID	对应于YDProduct.ProductID，唯一确定产品
	IntValue1	距离到期日的天数阈值

由于AccountID可以为全体账户或者指定账户两种，因此一共两个层级，他们优先级从高到低（高优先级覆盖低优先级的参数配置）分别是：

- 指定账户
- 对所有投资者

目前本风控规则由外部风控系统负责执行，因此投资者无法通过API获取具体风控参数值，也无法获知风控规则是否触发。

8.2.6. 期权系列基础风控

易达的传统风控规则中并不支持期权系列这一风控维度的设置，为了满足在期权系列维度风控的需要，我们提供了一系列基于期权系列的基础风控规则。与传统风控中的事前风控不同，基于期权系列的风控规则都是事后风控，也就是说无法保证不超过设定的风控阈值，因此，若需绝对不超过业务上实际需要的阈值，可以设置一个更强的风控阈值以避免极端条件下超过阈值太多。例如，业务上持仓量风控的阈值要求是100手，那么可以设置为90，留出了10手的超出空间。

下列风控规则中，使用的期权系列的描述方式是"期权产品代码:期权到期年份:期权到期月份"。

8.2.6.1. 期权系列持仓量风控

当该期权系列的多头总持仓大于等于多头限仓，或者空头总持仓大于等于空头限仓时，对该账户该期权系列的所有合约设置只平权限。如果之后通过平仓，使总持仓小于限仓了，本规则不会自动设置允许交易权限，但是如果管理员手工将其设置为允许交易权限，本系统也不会再次将其改为只平权限。

本风控规则的参数如下：

层级	字段	说明

层级	字段	说明
产品级	GeneralRiskParamType	固定为1005
	AccountID	空表示对所有投资者生效，否则表示用户的AccountID值
	ExtendedID	目前无法下发
	IntValue1	多头限仓
	IntValue2	空头限仓

由于AccountID可以为全体账户或者指定账户两种，因此一共两个层级，他们优先级从高到低（高优先级覆盖低优先级的参数配置）分别是：

- 指定账户
- 对所有投资者

目前本风控规则由外部风控系统负责执行，因此投资者无法通过API获取具体风控参数值，也无法获知风控规则是否触发。

8.2.6.2. 期权系列成交量风控

当该期权系列的总成交量大于等于最大成交量时，对该账户该期权系列的所有合约设置只平权限。

本风控规则的参数如下：

层级	字段	说明
产品级	GeneralRiskParamType	固定为1006
	AccountID	空表示对所有投资者生效，否则表示用户的AccountID值
	ExtendedID	目前无法下发
	IntValue1	最大成交量

由于AccountID可以为全体账户或者指定账户两种，因此一共两个层级，他们优先级从高到低（高优先级覆盖低优先级的参数配置）分别是：

- 指定账户
- 对所有投资者

目前本风控规则由外部风控系统负责执行，因此投资者无法通过API获取具体风控参数值，也无法获知风控规则是否触发。

8.2.6.3. 期权系列开仓量风控

当该期权系列的总买开仓量大于等于最大买开仓量，或者卖开仓量大于等于最大卖开仓量时，对该账户该期权系列的所有合约设置只平权限。

本风控规则的参数如下：

层级	字段	说明
产品级	GeneralRiskParamType	固定为1007
	AccountID	空表示对所有投资者生效，否则表示用户的AccountID值
	ExtendedID	目前无法下发
	IntValue1	最大买开仓量
	IntValue2	最大卖开仓量

由于AccountID可以为全体账户或者指定账户两种，因此一共两个层级，他们优先级从高到低（高优先级覆盖低优先级的参数配置）分别是：

- 指定账户
- 对所有投资者

目前本风控规则由外部风控系统负责执行，因此投资者无法通过API获取具体风控参数值，也无法获知风控规则是否触发。

9. 其他

9.1. 修改密码

盘中投资者可以通过API修改交易密码，修改后密码永久有效。调用方法如下：

```
virtual bool changePassword(const char *username,const char
*oldPassword,const char *newPassword)
```

密码的修改结果会通过以下回调返回，errorNo为0表示修改成功，其他值表示修改错误，通常情况是因为旧密码错误YD_ERROR_OldPasswordMismatch导致的。

```
virtual void notifyChangePassword(int errorNo)
```

9.2. 写日志

易达API会默认往可执行程序路径下的log目录写入日志，投资者可以借用该机制写入自己的日志信息，若找不到log目录则不会产生日志文件。

```
virtual void writeLog(const char *format,...)
```

该日志是按天拆分的，通常API在启动的时候会产生以下日志样例。

```
14:59:54 YD API start
14:59:54 version 1.108.36.33
14:59:54 build time Mar  3 2022 17:37:32
14:59:54 build version GCC 10.2.0
14:59:54 TCP trading server 0 connected
```